

# Self-Supervised Learning on Mobile Robots Using Acoustics, Vibration, and Visual Models to Build Rich Semantic Terrain Maps

Jacqueline Libby

CMU-RI-TR-19-82

December 16, 2019

The Robotics Institute  
School of Computer Science  
Carnegie Mellon University  
Pittsburgh, PA

**Thesis Committee:**

Anthony Stentz (chair)

Martial Hebert

David Wettergreen

Larry H. Matthies, *Jet Propulsion Laboratory*

*Submitted in partial fulfillment of the requirements  
for the degree of Doctor of Philosophy in Robotics.*

Copyright © 2019 Jacqueline Libby



## Abstract

Humans and robots would benefit from having rich semantic maps of the terrain in which they operate. Mobile robots equipped with sensors and perception software could build such maps as they navigate through a new environment. This information could then be used by humans or robots for better localization and path planning, as well as a variety of other tasks. However, it is hard to build good semantic maps without a great deal of human effort and robot time. Others have addressed this problem, but they do not provide a high level of semantic richness, and in some cases their approaches require extensive human data labeling and robot driving time.

We use a combination of better sensors and features, both proprioceptive and exteroceptive, and self-supervised learning to solve this problem. We enhance proprioception by exploring the use of new sensing modalities such as sound and vibration, and in turn we increase the number and variety of terrain types that can be estimated. We build a supervised proprioceptive multiclass model that predicts seven terrain classes. The proprioceptive predictions are then used as labels to train a self-supervised exteroceptive model from camera data. This exteroceptive model can then estimate those same terrain types more reliably in new environments. The exteroceptive semantic terrain predictions are spatially registered into a larger map of the surrounding environment. 3d point clouds from rolling/tilting lidar are used to register the proprioceptive and exteroceptive data, as well as to register the resulting exteroceptive predictions into the larger map. Our claim is that self-supervised learning makes the exteroception more reliable since it can be automatically retrained for new locations without human supervision. We conducted experiments to support this claim by collecting data sets from different geographical environments and then comparing classification accuracies. Our results show that our self-supervised learning approach is able to outperform state of the art supervised visual learning techniques.



## Acknowledgments

For our past work on the Gator platform, I would like to thank Scott Perry for help with field experiments and Don Salvadori for help with mechanical fabrication. For more recent work on the LAGR platform, I would like to thank Herman Herman, Jeffrey McMahill, Vladimir Altman, and James Ketterer for their development of the hardware and interface control. I would like to thank Dale Lord and Michael McGraw for their help with software development. I would like to thank Oliver Kroemer for helping me to incorporate the Variational Auto Encoder, and for the overall feedback he has given me in support of my research. Learning an unsupervised feature learning technique for robotic perception is an invaluable tool to have in my tool belt.

I would like to thank all of my committee members for their knowledge and insight in the field robotics and perception domains. Larry Matthies, my external committee member, has conducted extensive research in self-supervised learning, using proprioception to teach exteroception. He develops groundbreaking robots for NASA at the Jet Propulsion Laboratory, steering the course of computer vision and mobile robotics research in space. His insights have always been very useful, and his critiques of my work have helped me to refine and develop it along the way. Martial Hebert was my computer vision teacher early on in my studies at Carnegie Mellon, and I learned a tremendous amount in his class. His knowledge in computer vision has steered me along the right course for adding vision into my own research. During my time at Carnegie Mellon I have seen him advance to become the director of the Robotics Institute, and now to become the Dean of the School of Computer Science. But to me, he has always just been this awesome computer vision researcher who inspired me to delve into robotic perception techniques. David Wettergreen has been an inspiration in the domains of robotic perception, field robotics, and systems integration. He develops the next generation of space robots, and travels to the far ends of the earth to test and deploy these systems. He has given me extensive feedback to keep my research relevant, and I really value the conversations that we have had. During my time here, I have watched him advance to become the head of the PhD program. As the head of my program, he has helped me stay on track, and I thank him greatly for his guidance and direction.

In particular, I would like to thank my adviser, Tony Stentz, for continually guiding me in the right direction, and for providing me with his extensive wisdom in the domains of field robotics, mobile robotics, and systems integration. As former head of the National Robotics Engineering Center, he has always had a tremendous wealth of knowledge for building real robotics systems that actually work in harsh outdoor environments. Now at the Uber Advanced Technologies Group, he is continuing to push the envelope for the development of outdoor autonomous robotic vehicles in real-world applications. Tony has been an inspiration for how to think clearly and critically about building com-

plex robotic systems in the most practical way possible. He has stuck with me through the long haul, always with a positive attitude. He has taught me how to approach my research with grace and poise.

This work was conducted in part through collaborative participation in the Robotics Consortium sponsored by the U.S Army Research Laboratory under the Collaborative Technology Alliance Program, Cooperative Agreement W911NF-10-2-0016. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Problem Space . . . . .	2
1.3	Perception Techniques . . . . .	3
<b>2</b>	<b>Prior Work</b>	<b>7</b>
2.1	Learning with Proprioception on Mobile Robots . . . . .	7
2.2	Self-Supervised Learning with Proprioception on Mobile Robots . . . . .	9
2.2.1	A Direct Framework . . . . .	10
2.2.2	Two-step Frameworks . . . . .	12
2.3	Comparison of Approaches . . . . .	14
2.4	Thesis Problem . . . . .	19
2.5	Thesis Outline . . . . .	19
<b>3</b>	<b>Experimental Setup</b>	<b>21</b>
3.1	Gator Platform and Data Collection . . . . .	21
3.2	Proprioceptive Sensing Modalities . . . . .	25
3.3	LAGR Platform . . . . .	31
3.4	LAGR Data Collection . . . . .	35
<b>4</b>	<b>Self-Supervised Framework</b>	<b>43</b>
4.1	Data Registration . . . . .	44
4.2	Applying the General Framework To our System . . . . .	51
4.3	Four Main Experiments . . . . .	55
4.4	Thesis Problem Revisited . . . . .	60
<b>5</b>	<b>Proprioception Module</b>	<b>63</b>
5.1	Acoustics and Vibration Literature . . . . .	63
5.2	Data Overview and Hand Labeling . . . . .	64
5.3	Feature Extraction . . . . .	66
5.4	Classification . . . . .	69
5.5	Gator Experiments and Results . . . . .	70
5.6	LAGR Experiments and Results . . . . .	75
5.6.1	Learned Sensor Selection . . . . .	75

5.6.2	Test Results . . . . .	83
<b>6</b>	<b>Exteroception Module</b>	<b>87</b>
6.1	Unsupervised Feature Learning . . . . .	87
6.2	Classification . . . . .	90
6.3	Experiments and Results . . . . .	92
<b>7</b>	<b>Final Experiments and Results</b>	<b>103</b>
7.1	Proprioception . . . . .	103
7.2	Vision Floor . . . . .	105
7.3	Vision Ceiling . . . . .	107
7.4	Self-Supervision . . . . .	109
7.5	Analysis . . . . .	111
<b>8</b>	<b>Conclusion</b>	<b>115</b>
8.1	Contributions . . . . .	115
8.2	Future Work . . . . .	119
<b>9</b>	<b>Appendix</b>	<b>125</b>
9.1	Convolutional Variational Auto Encoder Images . . . . .	125
9.2	Self-Supervision Images . . . . .	140
9.2.1	Pixim Images for Previous Locales (sea2-11, wood1-7, wood1-8, bush4-1, tree2-1) . . . . .	140
9.2.2	Pixim Images for Current Locale (bramble1-1) . . . . .	148
9.2.3	Vision Floor Benchmark Training Labels . . . . .	152
9.2.4	Vision Floor Benchmark Test Predictions . . . . .	160
9.2.5	Self-Supervision Training Labels (snowball) . . . . .	164
9.2.6	Self-Supervision Test Predictions (snowball) . . . . .	167
9.2.7	Self-Supervision Training Labels (vt500-bumper, adxl-axle-up) . . . . .	171
9.2.8	Self-Supervision Test Predictions (vt500-bumper, adxl-axle-up) . . . . .	174
9.2.9	Vision Ceiling Benchmark Training Labels . . . . .	178
9.2.10	Vision Ceiling Benchmark Test Predictions . . . . .	181
	<b>Bibliography</b>	<b>185</b>



# List of Figures

1.1	Example of how a visual classifier can fail when detecting water . . . . .	5
2.1	General framework for self-supervision, where the teacher’s measurements are direct . . . . .	10
2.2	General two-step framework, where initial features are learned and then used to train additional features . . . . .	12
2.3	Example of an extension to the general frameworks . . . . .	14
3.1	The E-Gator platform . . . . .	22
3.2	Data collection environments for the <i>water</i> class . . . . .	23
3.3	Example of a data collection for the <i>hitting hard objects</i> class . . . . .	24
3.4	LAGR Platform . . . . .	32
3.5	Exteroceptive Sensors on LAGR Platform . . . . .	33
3.6	Proprioceptive Sensors on LAGR Platform, High Level . . . . .	33
3.7	Proprioceptive Sensors on LAGR Platform, Inside the Axle and Bumper Suites . . . . .	34
3.8	Data Collection Procedure . . . . .	37
4.1	General two-step framework (repeated for convenience) . . . . .	44
4.2	Coordinate Frame Block Diagram . . . . .	45
4.3	Robot at a Stopped Interval, Collecting Exteroceptive Data . . . . .	46
4.4	Robot During a Moving Interval, Collecting Proprioceptive Data . . . . .	46
4.5	Time Series Signal Representing an Instance of the Robot’s Interaction with a Particular Terrain Class . . . . .	48
4.6	A Pair of Images from the Left and Right Pixim Cameras . . . . .	49
4.7	A Pixim Pair from a Stopped Interval, with the 3-second Wobbler Point Cloud Projected onto Them . . . . .	49
4.8	Rectangular Footprints Projected onto the Images, Showing the Robot’s Path	50
4.9	Wobbler points that are Inside the Robot’s Path . . . . .	51
4.10	Wobbler points that are Outside the Robot’s Path . . . . .	51
4.11	Relationship between Theoretical Diagram and Actual Data . . . . .	52
4.12	Graphic for Inside the Robot’s Path . . . . .	54
4.13	Graphic for Outside the Robot’s Path . . . . .	54
4.14	Relationship between Theoretical Framework and Graphics Notation . . . . .	55

5.1	Spectrograms for each class. The x axis is time. The labeled terrain interaction sequences for each class are concatenated together in succession over time along the x axis. The colors show the percentage of spectral magnitude at each frequency, with red being the highest and blue being the lowest. . .	68
5.2	Balanced accuracies from acoustics experiments for the different feature/classifier combinations using just the front microphone . . . . .	72
5.3	Balanced accuracies from acoustics experiments with different ways of combining the microphone data . . . . .	73
6.1	Variational Auto Encoder From a High Level . . . . .	88
6.2	Robot Viewing Scene from Stopped Intervals Along Path . . . . .	90
6.3	Example of 10 Data Points for the Bramble Class . . . . .	90
6.4	Exteroceptive Data Inside and Outside the Robot's Path . . . . .	93
6.5	Example Training and Validation Data from the <i>tree2-1</i> locale, with the (pavement, treeBig) Binary Classifier . . . . .	100
8.1	Our current two-step self-supervised framework . . . . .	119
8.2	A hierarchical scheme to semantic labeling of terrain classes . . . . .	119
8.3	An unsupervised variant to the two-step framework . . . . .	121
9.1	100 Input Image Patches of the Grass Class . . . . .	126
9.2	100 Decoded Outputs of the Grass Class, corresponding to inputs in figure 9.1	127
9.3	100 Input Image Patches of the Grass-leaves Class . . . . .	128
9.4	100 Decoded Outputs of the Grass-leaves Class, corresponding to inputs in figure 9.3 . . . . .	129
9.5	100 Input Image Patches of the Pavement Class . . . . .	130
9.6	100 Decoded Outputs of the Pavement Class, corresponding to inputs in figure 9.5 . . . . .	131
9.7	100 Input Image Patches of the Soft Vegetation Class . . . . .	132
9.8	100 Decoded Outputs of the Soft Vegetation Class, corresponding to inputs in figure 9.7 . . . . .	133
9.9	100 Input Image Patches of the Bramble Class . . . . .	134
9.10	100 Decoded Outputs of the Bramble Class, corresponding to inputs in figure 9.9 . . . . .	135
9.11	100 Input Image Patches of the Bush Class . . . . .	136
9.12	100 Decoded Outputs of the Bush Class, corresponding to inputs in figure 9.11	137
9.13	100 Input Image Patches of the Tree Class . . . . .	138
9.14	100 Decoded Outputs of the Tree Class, corresponding to inputs in figure 9.13	139
9.15	Pixim Images for Previous Locales (sea2-11, wood1-7, wood1-8, bush4-1, tree2-1), Image 1 . . . . .	140
9.16	Pixim Images for Previous Locales (sea2-11, wood1-7, wood1-8, bush4-1, tree2-1), Continued, Images 2 - 3 . . . . .	141
9.17	Pixim Images for Previous Locales (sea2-11, wood1-7, wood1-8, bush4-1, tree2-1), Continued, Images 4 - 5 . . . . .	142

9.18 Pixim Images for Previous Locales (sea2-11, wood1-7, wood1-8, bush4-1, tree2-1), Continued, Images 6 - 7 . . . . .	143
9.19 Pixim Images for Previous Locales (sea2-11, wood1-7, wood1-8, bush4-1, tree2-1), Continued, Images 8 - 9 . . . . .	144
9.20 Pixim Images for Previous Locales (sea2-11, wood1-7, wood1-8, bush4-1, tree2-1), Continued, Images 10 - 11 . . . . .	145
9.21 Pixim Images for Previous Locales (sea2-11, wood1-7, wood1-8, bush4-1, tree2-1), Continued, Images 12 - 13 . . . . .	146
9.22 Pixim Images for Previous Locales (sea2-11, wood1-7, wood1-8, bush4-1, tree2-1), Continued, Images 14 - 15 . . . . .	147
9.23 Pixim Images for Current Locale (bramble1-1), Image 1 . . . . .	148
9.24 Pixim Images for Current Locale (bramble1-1), Continued, Images 2 - 3 . . . . .	149
9.25 Pixim Images for Current Locale (bramble1-1), Continued, Images 4 - 5 . . . . .	150
9.26 Vision Floor Benchmark Training Labels, Images 1 - 2 . . . . .	152
9.27 Vision Floor Benchmark Training Labels, Continued, Images 3 - 4 . . . . .	153
9.28 Vision Floor Benchmark Training Labels, Continued, Images 5 - 6 . . . . .	154
9.29 Vision Floor Benchmark Training Labels, Continued, Images 7 - 8 . . . . .	155
9.30 Vision Floor Benchmark Training Labels, Continued, Images 9 - 10 . . . . .	156
9.31 Vision Floor Benchmark Training Labels, Continued, Images 11 - 12 . . . . .	157
9.32 Vision Floor Benchmark Training Labels, Continued, Images 13 - 14 . . . . .	158
9.33 Vision Floor Benchmark Training Labels, Continued, Image 15 . . . . .	159
9.34 Vision Floor Benchmark Test Predictions, Images 1 - 2 . . . . .	160
9.35 Vision Floor Benchmark Test Predictions, Continued, Images 3 - 4 . . . . .	161
9.36 Vision Floor Benchmark Test Predictions, Continued, Image 5 . . . . .	162
9.37 Self-Supervision Training Labels (snowball), Images 1 - 2 . . . . .	164
9.38 Self-Supervision Training Labels (snowball), Continued, Images 3 - 4 . . . . .	165
9.39 Self-Supervision Training Labels (snowball), Continued, Image 5 . . . . .	166
9.40 Self-Supervision Test Predictions (snowball), Images 1 - 2 . . . . .	167
9.41 Self-Supervision Test Predictions (snowball), Continued, Images 3 - 4 . . . . .	168
9.42 Self-Supervision Test Predictions (snowball), Continued, Image 5 . . . . .	169
9.43 Self-Supervision Training Labels (vt500-bumper, adxl-axle-up), Images 1 - 2 . . . . .	171
9.44 Self-Supervision Training Labels (vt500-bumper, adxl-axle-up), Continued, Images 3 - 4 . . . . .	172
9.45 Self-Supervision Training Labels (vt500-bumper, adxl-axle-up), Continued, Image 5 . . . . .	173
9.46 Self-Supervision Test Predictions (vt500-bumper, adxl-axle-up), Images 1 - 2 . . . . .	174
9.47 Self-Supervision Test Predictions (vt500-bumper, adxl-axle-up), Continued, Images 3 - 4 . . . . .	175
9.48 Self-Supervision Test Predictions (vt500-bumper, adxl-axle-up), Continued, Image 5 . . . . .	176
9.49 Vision Ceiling Benchmark Training Labels, Images 1 - 2 . . . . .	178
9.50 Vision Ceiling Benchmark Training Labels, Continued, Images 3 - 4 . . . . .	179
9.51 Vision Ceiling Benchmark Training Labels, Continued, Image 5 . . . . .	180
9.52 Vision Ceiling Benchmark Test Predictions, Images 1 - 2 . . . . .	181

9.53	Vision Ceiling Benchmark Test Predictions, Continued, Images 3 - 4 . . . .	182
9.54	Vision Ceiling Benchmark Test Predictions, Continued, Image 5 . . . . .	183

# List of Tables

2.1	Comparison of properties estimated from supervised learning algorithms that use proprioceptive sensors on mobile robots . . . . .	9
2.2	Comparison of self-supervised learning implementations on mobile robots, where the teacher was proprioceptive . . . . .	15
2.3	Comparison of proprioceptive learning approaches on mobile robots, and how they address the three dimensions of the problem space . . . . .	18
3.1	Number of Data Points for Each Class on Gator Platform . . . . .	25
3.2	Comparison of proprioceptive measurements and semantic richness . . . . .	26
3.3	Comparison of sensors used for measuring vibration . . . . .	27
3.4	A list of the sound and vibration sensors that we used in our experiments.	30
3.5	Locales and Terrain Classes in Each Locale . . . . .	38
3.6	Number of Data Points for Proprioceptive Data . . . . .	40
3.7	Number of Data Points for Exteroceptive Data . . . . .	41
4.1	Proprioception Overview . . . . .	56
4.2	Self-Supervision Overview . . . . .	57
4.3	Vision Floor Overview . . . . .	59
4.4	Vision Ceiling Overview . . . . .	60
5.1	Number of training and test data points used in Gator acoustics experiments	71
5.2	Normalized confusion matrix from the acoustics trial with the best average accuracy . . . . .	73
5.3	Classification accuracies from experiments using both microphone and vibration data . . . . .	75
5.4	List of 19 Proprioceptive Sensor Signals . . . . .	76
5.5	Training and Validation Locales for Proprioceptive Sensor Selection . . . . .	78
5.6	Number of Training and Validation Data Points for Proprioceptive Sensor Selection . . . . .	79
5.7	Confusion Matrix for Validation Data (locale veg2-1), Sensor = (snowball)	80
5.8	Evaluation Metrics for Validation Data (locale veg2-1), Sensor = (snowball)	81
5.9	Top 10 Sensor Combinations and Accuracies . . . . .	82
5.10	Confusion Matrix for Validation Data (locale veg2-1), Sensor Combination = (vt500-bumper, adxl-axle-up) . . . . .	83

5.11	Evaluation Metrics for Validation Data (locale veg2-1), Sensor Combination = (vt500-bumper, adxl-axle-up) . . . . .	83
5.12	Training and Test Locales for Proprioception Final Results . . . . .	84
5.13	Number of Training and Test Data Points for Proprioception Final Results	84
5.14	Confusion Matrix for Test Data (locale bramble1-1), Sensor Combination = (vt500-bumper, adxl-axle-up) . . . . .	85
5.15	Confusion Matrix for Test Data (locale bramble1-1), Sensor = (snowball) .	85
5.16	Recall Comparison of Base Sensor (snowball) to Learned Sensor Combination (vt500-bumper, adxl-axle-up) . . . . .	86
5.17	Normalized Precision Comparison of Base Sensor (snowball) to Learned Sensor Combination (vt500-bumper, adxl-axle-up) . . . . .	86
6.1	Locales and Terrain Classes in Each Locale Used for VAE . . . . .	89
6.2	Locales and Terrain Classes in Each Locale Used for Exteroceptive Feature Comparison . . . . .	92
6.3	Number of Training and Validation Data Points for Exteroceptive Ceiling Experiments . . . . .	93
6.4	List of Feature/Classifier Combinations on which We Run Experiments . .	95
6.5	Recall Comparison, Validation Data . . . . .	97
6.6	Precision Comparison, Validation Data . . . . .	98
7.1	Training and Test Locales . . . . .	103
7.2	Proprioception Overview . . . . .	104
7.3	Number of Training and Test Data Points for Proprioception Final Results	104
7.4	Confusion Matrix for Test Data (locale bramble1-1), Sensor Combination = (vt500-bumper, adxl-axle-up) . . . . .	105
7.5	Vision Floor Overview . . . . .	106
7.6	Number of Training and Test Data Points for Vision Floor Final Results .	107
7.7	Vision Floor Benchmark Confusion Matrix for Test Data (locale bramble1-1)	107
7.8	Vision Ceiling Overview . . . . .	108
7.9	Number of Training and Test Data Points for Vision Ceiling Final Results	108
7.10	Vision Ceiling Benchmark Confusion Matrix for Test Data (locale bramble1-1, outside robot's path) . . . . .	109
7.11	Self-Supervision Overview . . . . .	110
7.12	Number of Training and Test Data Points for Self-Supervision Final Results	111
7.13	Self-Supervision Confusion Matrix for Test Data (locale <i>bramble1-1</i> , outside robot's path), using proprioception with (vt500-bumper, adxl-axle-up) sensors	111
7.14	Recall Comparison . . . . .	112
7.15	Normalized Precision Comparison . . . . .	113
7.16	Normalized Precision Comparison Without Tree . . . . .	114

# Chapter 1

## Introduction

### 1.1 Motivation

As robots become more sophisticated and capable of handling complex tasks, their perception systems must become more advanced in order to more fully comprehend their environment. One way to comprehend the environment is to label objects or regions with semantic information. This information could then be spatially registered into a map, enabling the robot to build a semantically rich model of the world around it. Individual robots, teams of robots, and mixed teams of robots and people could benefit from having a shared model of the world. This would allow them to work on tasks together and communicate based on this shared understanding. *Mobile* robots must be able to navigate complex, unknown environments. In off-road scenarios, a mobile robot must be able to detect and avoid hazardous terrain. Its perception system must be accurate and robust to failure. Mapped-out models of the environment could also be used in monitoring scenarios, where the map is tracked over time and changes in the environment are recorded.

A detailed understanding of the environment allows a mobile robot to be more effective in path planning, motion planning and control. Path planning requires a cost map so that the robot can avoid high cost, or hazardous, situations. Perceiving the semantics of a region could aid in determining a cost for that region. For example, a robot might attribute a high cost to certain terrain types such as “water” or a “bush” or a “cliff” or a “tree”. Understanding different ground types, such as “gravel”, “sand”, and “pavement” could also help with building cost maps. For instance, the robot might want to avoid sand because there is a higher chance of getting stuck. Even if the robot is not avoiding a high cost region, it might want to choose a certain control mode for more efficient vehicle mobility in that region, such as choosing the appropriate maximum speeds, turning angles, and accelerations (Coyle and Collins, [11]). The same can be true for above-ground object identification. For instance, the robot might decide that it can drive over certain types of bushes, so categorizing the bush would allow for the use of a more sophisticated dynamics-based motion planner (Bajracharya et al. [7]).

A richly detailed world map could be useful for robot navigation, just as it is for human navigation. Google maps lets a human navigate a path to a new location. When a simple

map is insufficient, Google Street View allows one to match an image of the location with what one sees. This takes advantage of the sophisticated semantic models that humans use to recognize buildings, glass storefronts, park settings, etc. Using Google Street View to match the semantic information in an image to a particular location is an example of a human *localizing* himself or herself within a map using landmarks. This is similar to the way a robot would use a landmark localization filter. The hardest part of such a filter (often not even stated in the algorithm) is the data association step, which is the process of distinguishing between different landmarks. The richer the models for the landmarks, the easier it is to distinguish them from each other.

Many specific applications for mobile robots would benefit from rich semantic maps. Exploration robots, such as the Mars Rovers, could use this information to automatically understand what they are experiencing without having to ask a human teleoperator for help. There can be very long delays with transmitting information back and forth from Mars to Earth, so minimizing the amount of human labeling is very pertinent in this situation. In search and rescue scenarios, a robot must be able to navigate an unknown environment to carry supplies or to rescue a soldier from a burning building or an open field. Detecting landmines is another example. Often the material within the buried landmine will change the soil chemistry around it and affect the type of vegetation that grows there. Learning what types of vegetation grow in these regions and learning to detect these vegetation models are both active areas of research in this domain. Handling disaster scenarios, such as the Fukushima nuclear meltdown or the Gulf of Mexico oil spill, is another important application for mobile robotics (both on land and underwater). These vehicles must be able to navigate and monitor these environments, which involves understanding the semantic details of what is around them.

## 1.2 Problem Space

The question then becomes how to formulate a problem space that we can use to work towards this overarching goal of a semantically richer world model. We assume that these models are too complex for the most part to be derived analytically, so we turn to machine learning as a solution for developing these models. We can break this challenge up into three dimensions that we would like to optimize:

1. Increase the level of semantic richness these models can provide
2. Decrease the amount of human effort required to teach these models to robots
3. Decrease the amount of robot effort required to learn these models

The first dimension, how *semantically rich* these models can get, is a question of how much the learning algorithms can estimate. We would like to increase the *number* and *variety* of categories (or semantic labels) that can be distinguished.

The second dimension, how much *human effort* is required, is a measure of how much training data must be labeled by humans in order for the models to be successfully learned. As mentioned above, there can be variation within each terrain category, and sometimes this variation can change from one location to another. So the question becomes how to



decrease the number of locations where data must be human-labeled in order for the models to generalize to future locations. We want to achieve this while still working towards the goal in the first dimension: increasing the semantic richness of the model.

The third dimension, how much *robot effort* is required, is a measure of how much work the robot must perform in order to gather the data. This can be directly correlated with how close the robot must come to the source of the data; namely, the environment around it. The closer the robot must come to the source of the data, the more work it will take to cover a given spatial region. However, this data will have higher spatial resolution and more sensing modalities can be effective. An example of very low-effort data collection would be satellite imagery, where any one image covers a vast amount of terrain. However, each pixel of the image is at a very low spatial resolution. A ground vehicle driving through that same terrain could collect data from a variety of sensors that work at close distances (cameras, range sensors, radar, etc.). The vehicle would have to drive a longer path to cover the same region as the satellite, but would attain much higher resolution data. A ground vehicle could also *interact* with the terrain, either by driving over it, bumping into it, or manipulating it with a robot arm. Interaction would allow for an entirely new set of sensing modalities with new types of information, but would require even more work: the robot would have to make physical contact with every patch of terrain for which it wants to acquire data, and the interaction at each terrain patch would require more effort than just visualizing that patch. The question becomes how to decrease the amount of work the robot must perform, given the constraint of covering as large of a spatial region as possible.

### 1.3 Perception Techniques

This leads us into an exploration of different robotic perception techniques and what they can afford us across the three dimensions of the problem space listed in section 1.2. Robotic perception can be broken down into *exteroception* and *proprioception*. Exteroception is the measurement of the environment external to the robot. Examples include appearance features coming from camera imagery and geometry features coming from range data. Proprioception is the measurement of the internal state of the robot. Examples of internal properties that a mobile robot can sense include the angle of its steering wheel or how fast its ground wheels are spinning. However, proprioceptive sensing can also be used to measure the external environment by *interacting* with the environment and then measuring the effect that the interaction has internally on the robot. For instance, as a robot drives over a surface (thereby interacting with that surface), vibration sensors can pick up signatures of the ground type. Another example is if a robot attempts to drive into an object (thereby interacting with that object), motion sensors can determine the compliance of that object; in other words, whether the robot can drive over the object or not. For our goal of building a semantically rich world model, we are primarily interested in understanding the external environment; thus we can look at proprioceptive techniques used for interactions with the environment as well as traditional exteroceptive techniques.

Exteroceptive features such as appearance and geometry are powerful tools for classifying terrain in some region of space around a robot. Appearance features such as color and

texture can be obtained from camera imagery. Geometric features can be obtained from range data coming from sensors such as stereo cameras or lidar. However, there can be an extensive amount of variation within these visual features, which causes problems for classification algorithms. Take the example of distinguishing between a rock and a bush. This seems like it should be a relatively easy task for a visual classification algorithm, but when presented with enough examples from different environments, the variation makes the problem intractable. Color coming from camera imagery can fail because both rocks and bushes can be many different colors. Sometimes rocks can be covered with moss, making them appear green like a bush. When bushes lose their leaves, they can appear gray or brown like a rock. Furthermore, outdoor lighting variations can make color very tricky to use. Shape coming from range data can fail because rocks and bushes can both be many different shapes and sizes. They can both be pointed or round, and large or small. So this example demonstrates that both color (an appearance feature) and shape (a geometric feature) can fail as features to learn a distinguishing model.

Another example of a terrain type with significant visual variation is *water*. Figure 1.1 shows a robotic vehicle in two test locations (used in our early work, see section 3.1). On the left, the vehicle is driving through shallow murky water, and on the right, it is driving over a gravel road. It is clear that geometry cannot be used to distinguish between the two surfaces since they are both flat, but it seems like appearance features such as color or texture could be used. However, the water is murky and therefore not much different in color than the gravel road. Even if a slight difference in color between the two surfaces can be discerned, this would not generalize well to new environments or new lighting conditions in the same environment. Water can also be covered with vegetation, similar to the moss-covered rock in the previous example, which again makes it hard for color or texture features to be used. The water’s surface can also be calm or rough. The roughness could come from wind ripples, waves, or if the water is part of a running stream. Variations in the roughness can make it hard to use texture as a reliable feature. Rankin et al. classified water by using *reflection* as an appearance feature [30]. This involved identifying terrain patches in the surrounding region and then finding their reflections on the water’s surface. Using reflection features requires a great deal of spatial integrity and can fail if the surrounding terrain patches creating the reflection are located outside the boundaries of the image, or if there are occlusions that are not accounted for. Furthermore, the reflections can be lost or degraded from many of the environmental effects discussed above, such as murkiness, vegetation coverage, or surface roughness.

Proprioception used for interaction with the environment can measure different types of properties than what is provided by exteroception; hence it can be used to compensate in areas where exteroception fails. As a robot drives over a surface, vibration features can give information about the ground type. If a robot drives into an object, the ability of the robot to compress that object gives information about the object’s compliance. These are both examples where the robot is learning something about the material consistency of the environment, separate from its geometry or appearance. Take the previous example of distinguishing between a rock and a bush. Although the appearance and shape may vary, the compliance property of these objects would remain mostly constant.

In the above examples, the exteroceptive features would vary, while the propriocep-



(a) Vehicle in murky water



(b) Vehicle on gravel road

Figure 1.1: Example of how a visual classifier can fail when detecting water. (a) shows a robotic vehicle in murky water. (b) shows the vehicle on a gravel road. These surfaces are somewhat similar in appearance, and many factors could change their appearance to the point where optical sensing would not be enough to distinguish between the two terrain types.

tive features would remain consistent. The hypothesis then is that proprioceptive sensing would allow a robot to more readily learn a general model for the world. Since the model generalizes better to new environments, it would take less *human effort* to teach the model to the robot. Perhaps the model could be hardwired in, or perhaps there is some learning that is involved, but the human labels required for this learning only have to come from a few environments, because these environments would generalize well to new ones.

Although proprioception can be more consistent than exteroception, interaction requires more *robot effort*. The robot can only measure the terrain with which it interacts, so it would have to drive over every patch of terrain that will be registered in its map. Furthermore, the interaction itself takes more effort than just collecting visual data. Exteroceptive sensing can measure anything within view, registering information out into the space around it up to the limit of the sensing resolution. So we can view proprioception and exteroception as complementary. Note that the advantages and disadvantages of each directly correspond to the second and third dimensions of the problem space as formulated in section 1.2; namely, how much *human effort* and *robot effort* are needed. We summarize these advantages and disadvantages with the following bullet points:

- Proprioceptive sensing
  - Advantage: features can describe properties of the world more consistently across environments (*lower human effort*)
  - Disadvantage: it is harder to collect the data because the robot must interact with the environment (*higher robot effort*)
- Exteroceptive sensing
  - Advantage: data collection is easier because features can be registered into the surrounding space (*lower robot effort*)
  - Disadvantage: the features can have too much variation and therefore training has to be repeated in new locations (*higher human effort*)

*Self-supervised* learning is one way to combine these sensing modalities, leveraging the strengths of each. In *supervised* learning, the human tags the data with values for the property to estimate. The teacher is the set of human labels and the student is the set of features from the data. In *self-supervised* learning, one sensing modality acts as the teacher while another sensing modality acts as the student. Predictions from the first modality provide ground truth values. Features from the second modality use these values to learn a model for making the same predictions.

There is a small body of research in self-supervised learning on mobile robots where proprioception acts as the teacher and exteroception acts as the student. The proprioception is the teacher since its features are more consistent across environments (which means *lower human effort*). It teaches the exteroception how to recognize the terrain, and this can be done automatically for any new environment. For each new environment where this automatic training happens, some minimal amount of proprioceptive data is acquired through vehicle-terrain interaction. Then once the exteroception learns the model, it can map its predictions spatially outward onto a larger amount of terrain (which means *lower robot effort*).

By using proprioception as the teacher, the *semantic richness* of the world model is limited to what the proprioception is capable of estimating (which is the first dimension of the problem space). The question then becomes how many properties can be estimated by proprioception; in other words, how far can the limits of proprioceptive sensing modalities be pushed in order to maximize the richness of the world model?

In chapter 2 we examine prior work that has made strides in these domains. Using proprioception on mobile robots as an interactive sensing modality to understand the external environment is a relatively unexplored research direction, and section 2.1 discusses work in this area. Furthermore, using interactive proprioception to teach exteroception on mobile robots in a self-supervised learning scheme is also a relatively unexplored area, and section 2.2 discusses research here. As we look at this prior work, we define some frameworks for contextualizing them. We then use these frameworks to define our problem more specifically in section 2.4, and discuss how our approach is unique from prior work. Thus we transform the problem space presented in section 1.2 into a more well-defined thesis problem. Once the thesis problem has been defined, we conclude the chapter with an outline for the remainder of the document in section 2.5.

# Chapter 2

## Prior Work

### 2.1 Learning with Proprioception on Mobile Robots

This section surveys a small body of work on mobile robots where proprioceptive sensing was used to learn about the environment through interaction. As we discuss in the introduction, proprioceptive sensing is the ability for a robot to sense its internal state. Although the surrounding environment is external to the robot, the robot can interact with the environment and then proprioceptively sense its own internal response to that environment. All of the following examples used supervised learning to train features from proprioceptive sensors in order to estimate properties pertaining to the external environment.

Weiss et al. performed vibration-based classification to distinguish between seven ground types: vinyl floor, asphalt, gravel, grass, paving, clay court, and no motion [37]. They extracted features from the z-value of an accelerometer that is part of an Attitude Heading and Reference System (AHRS). The AHRS was strapped to a wheeled cart that was manually pushed over different terrain types, with the intent to later collect these signals from a mobile robot.

Coyle and Collins also performed vibration-based classification to distinguish between seven ground types: beach sand, packed clay, regular grass, tall grass, loose gravel, packed gravel, and asphalt [11]. They used the z-value, roll rate, pitch rate, and speed reported by an Inertial Measurement Unit (IMU) on-board a mobile robot.

Ojeda et al. performed terrain classification for five ground types: gravel, grass, sand, pavement, and dirt [25]. They experimented with a variety of different sensors, including microphones, gyroscopes, accelerometers, encoders, motor current, voltage sensors, and downward-facing ultrasonics and infrared. They used a separate classifier for each sensing modality and analyzed which modalities worked best for specific terrains, but did not attempt to combine the results. They found that gyroscopes worked best for gravel, pavement and dirt; motor current worked best for sand; and microphones worked best for grass.

Stavens et al. used imitation learning to model a velocity controller for a high-speed, off-road vehicle [34]. Shock measurements from an IMU as well as geometric terrain features from ladar were used to model the roughness of the ground. A human drove through some

example terrains, and then imitation learning was used to understand the ways in which the human accelerates on different ground types. Stanford’s 2005 DARPA grand challenge platform used this algorithm for some of its speed control.

In our 2012 ICRA paper [22], we performed acoustics-based classification to distinguish between five terrain types: grass, pavement, gravel, water and hard objects. We experimented with various feature extraction techniques fed into a multiclass classifier built with Support Vector Machines within a *one-vs-one* graph. To the best of the author’s knowledge, we were the first ones to classify a variety of ground and above-ground types using only sound.

Others extended our ICRA paper with acoustics-based terrain classification, but only on ground terrain classes. Christie and Kottege built a multiclass classifier with seven ground terrain classes: carpet, concrete, grass, mulch, gravel, tile and asphalt [10]. They extracted features from a microphone signal and fed these into a multiclass SVM.

Zhao et al. built a multiclass classifier with six ground terrain classes: brick, asphalt, grass, firm soil, gravel and soft soil [42]. They used four microphones and one vibration sensor. They used feature selection techniques from Relief and mRMR algorithms, building classifiers for each sensor signal separately. They then fused the predictions.

Yu and Lee used vibration data to predict one of five ground classes: cork, tile, corrugated cardboard, gravel and artificial turf [41]. They used a Bayesian random field to infer the ground type, and then together with the robot path, they used a Conditional Random Field to make predictions about the surrounding terrain.

Valada et al. built a multiclass classifier with nine ground terrain classes: asphalt, mowed grass, thick grass, pavement, cobblestone, off-road ground (dirt), wood, linoleum and carpet [35]. They fed the spectrogram images of microphone signals through Convolutional Neural Networks, comparing these results to approaches that extract features from the time series signals.

Table 2.1 summarizes the implementations discussed above for supervised learning on mobile robots that use proprioceptive sensing, excluding our work. Most of this body of work focused on classifying the material properties of the ground, except for Stavens et al. [34] who learned velocity control parameters. In all cases, the properties considered were constrained to properties of the ground, as opposed to objects or obstacles above ground such as rocks, trees, bushes, and man-made objects. Using proprioception to estimate the material properties of above-ground terrain types is an unexplored research direction, and we address this more definitively in section 2.3.

Authors	Properties Estimated
Stavens et al. [34]	Velocity control parameters
Ojeda et al. [25]	Gravel, grass, sand, pavement, dirt
Weiss et al. [37]	Vinyl floor, asphalt, gravel, grass, paving, clay court, no motion
Coyle, Collins [11]	Beach sand, packed clay, regular grass, tall grass, loose gravel, packed gravel, asphalt
Christie, Kottege [10]	Carpet, concrete, grass, mulch, gravel, tile, asphalt
Zhao et al. [42]	Brick, asphalt, grass, firm soil, gravel, soft soil
Yu, Lee [41]	Cork, tile, corrugated cardboard, gravel, artificial turf
Valada et al. [35]	asphalt, mowed grass, thick grass, pavement, cobblestone, off-road ground (dirt), wood, linoleum, carpet

Table 2.1: Comparison of properties estimated from supervised learning algorithms that use proprioceptive sensors on mobile robots

## 2.2 Self-Supervised Learning with Proprioception on Mobile Robots

We focus here on a small body of work in self-supervised learning where the teacher uses proprioception to estimate properties about the external environment through interaction with that environment, and the student uses exteroception to learn a model for estimating those same properties. In particular, the student is near-field exteroceptive sensing, which involves looking at a small local region of space around the robot (usually on the scale of about 10 to 20 meters). A complementary relationship exists between these two modalities, as discussed previously, which corresponds to the second and third dimensions of the problem space that we introduce in section 1.2. Proprioceptive sensing can only label what it comes into contact with (high robot effort), but it can provide features that are more consistent across different environments (low human effort). Near-field exteroceptive sensing can label anything in a local map region (low robot effort), but these labels do not generalize as well to new environments and are therefore less reliable (high human effort).

Localization and mapping within a relative local region is needed to spatially register the proprioceptive and exteroceptive information. For instance, imagine that at time  $t_1$  the robot is located at position  $a$ , and at that instant sees some patch of terrain in front of it, which is located at position  $b$ . Then the robot will move forward and attempt to drive over  $b$ , interacting with that terrain patch at time  $t_2$ . The robot now has exteroceptive information about  $b$  from time  $t_1$ , as well as proprioceptive information about  $b$  from time  $t_2$ . But in order to fuse this information, it needs to register both locations  $a$  and  $b$  within

a local map and then localize itself as it moves from  $a$  to  $b$ .

### 2.2.1 A Direct Framework

We have defined self-supervised learning as a situation where one sensing modality provides labels and a second sensing modality learns a model from these labels. The question, though, is how the the first sensing modality can provide the labels. This section discusses the most straightforward scheme for answering this question, as well as examples of prior work that used this scheme. In this framework, the first sensing modality has an analytical model that turns its measurements into property values. This model is analytical because it does not have to be learned from training data. It can be written as a function in a program before any experimentation is conducted. The second sensing modality then learns a model by using predictions from the first modality as training labels.

This framework is graphically depicted in Fig. 2.1. The block on the upper left is the teacher, the block on the upper right is the student, and the block on the bottom is the property being estimated. The solid arrow going from the teacher to the property indicates that the teacher’s measurement can be turned into a property value directly, without any learning. The dotted arrow coming out of the student indicates that the student must learn a model in order to predict the property values, using training data from the teacher’s labels. Proprioception and exteroception are written in green above the teacher and student, respectively, to highlight that we are looking at this subset of scenarios. The rest of this section discusses examples of prior work that fit into this framework.

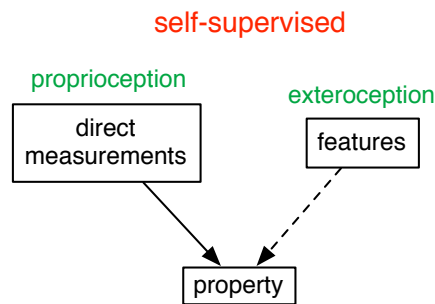


Figure 2.1: General framework for self-supervision, where the teacher’s measurements can directly be turned into property values

Stavens and Thrun estimated terrain roughness for an autonomous off-road vehicle [33]. (Note that this is separate from their work in supervised learning for a velocity controller that we discuss in section 2.1.) The teacher (top left block) was a set of shock measurements coming from an IMU on-board the vehicle. The shock measurements were turned into *ruggedness* coefficients that characterized the property being estimated (bottom block), which was the roughness of the ground. The student (top right block) was a set of features coming from a 3d point cloud generated by ladar. A regression model was learned, where



the input was the set of ladar features, and the output was the roughness of the ground. The final regression model was then thresholded at some empirical value to output a binary label for the ground: rough or not rough.

Kim et al. estimated terrain traversability for a LAGR platform [19]. (The LAGR platform was initially developed for the DARPA program Learning Applied to Ground Robots. A modified version of it is the platform for our work in this thesis, as discussed in chapter 3.3.) The property estimated was a binary label for whether the terrain was traversable or not. The teacher was a direct label for this property, which was determined by a combination of measurements from the IMU, the bumper, and the motor current. The LAGR would attempt to drive over objects in the terrain, and the sensors would provide information about whether or not the object in front of the robot had stopped the robot from moving. The bumper could tell the LAGR if it hit something hard. The motor current could tell it whether its wheels were spinning freely because the object was stopping it from moving forward. The IMU could tell it whether its velocity was zero or not. The student was a set of texture features from a stereo camera pair. A model was then learned that turned the camera features into a traversability label. The data sets consisted mostly of tall grass, logs and trees (tall grass being traversable, whereas logs and trees are not.) The learned model allowed the robot to recognize the appearance of these objects as traversable or not traversable. The underlying motivation is that exteroceptively perceiving the height of an object is not enough to learn the model, since grass can be just as tall as a log, for instance. Kim et al. called the property that they are estimating *terrain traversability*, but we will more specifically refer to this as *terrain compliance*, since traversability covers a wider set of circumstances. Here, they were looking specifically at determining that grass *is* compliant, while logs and trees are *not*.

Another example of estimating terrain compliance as a binary value is given in Howard et al. [17]. This work was again conducted on a LAGR platform. The interactive method was similar: the LAGR saw an object in front of it and attempted to drive over it. The teacher was a direct label coming from the bumper (which told the robot whether it had hit an object that was traversable or not). The student features are more complex than in the previous example because they use a variety of appearance features from the cameras as well as range features from the stereo camera pair. The authors extend this work in Bajracharya et al. [7], replacing the teacher with measurements from visual odometry instead of bumpers. (Note that in both of these examples, this self-supervised step was the first step in a multi-layered self-supervised framework which is not discussed here.)

Ott and Ramos [27] extend the work of Howard et al. The teacher was again a direct label from the bumper to determine whether an object is traversable or not. The student was appearance features from cameras. Their work is interesting because they perform unsupervised clustering on the appearance data, and then label each cluster with the proprioceptive traversability label.

Another example of estimating terrain compliance is given in Wellington and Stentz, but this time the property was estimated as a continuous value instead of a binary value [38]. This was one of the first examples of self-supervised learning being demonstrated on a mobile robot. The platform used was a robotic tractor with a very accurate 6 Degree Of Freedom (6-DOF) pose estimate from a filter that used a sophisticated suite of sensors,

including GPS, gyroscopes, Doppler radar, encoders and stereo cameras. Their goal was to estimate how compliant the vegetation was in front of the tractor. They estimated this by seeing how the height of the rear wheels changed as the robot drove over the vegetation. The teacher was the set of wheel height measurements coming from the pose data. The student was a set of range features coming from lidar. A regression model was learned that took as input the lidar features and the output was a height prediction. They could then predict the terrain compliance of the surrounding area from the lidar data. This helped them with other estimation, control and planning tasks on the system.

### 2.2.2 Two-step Frameworks

The previous section discussed a self-supervised framework where the teaching sensing modality has an analytical model for turning its measurements into property predictions. In this section, we discuss cases where the teaching modality does not have an analytical model, and so it must first learn one before teaching the student how to learn a model. In other words, the teacher must first be taught. This adds an additional preliminary step to the process. The key idea is that it is *easier* for the teacher to learn a model than it is for the student to learn a model. Figure 2.2 depicts a general framework where this preliminary step is a supervised learning module.

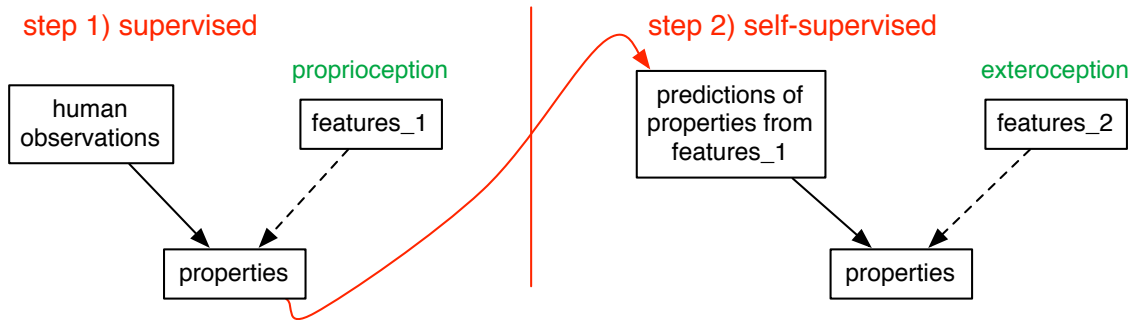


Figure 2.2: General two-step self-supervised framework. In the first step, supervised learning is used to train one set of features. In the second step, the model trained from the first set is used to train a second set of features. This framework is useful when it is easier to train the first set of features with human supervision than it would be to train the second set.

This framework contains an initial supervised step (shown on the left) and a secondary self-supervised step (shown on the right). In each step, solid black arrows denote teachers and dotted black arrows denote students. In the first supervised step, a human provides labels for training data that is used to build a model for the first sensing modality. In the second self-supervised step, the model built for the first sensing modality provides predictions on additional data, and this additional labeled data is used to train the second sensing modality. The first step is *easier* than the second step because the first model

requires fewer labels. The human only has to provide labels for a small set of data; once the teacher is trained, it can go on to teach the student on a larger set of data.

Proprioception and exteroception are written in green above the feature sets in figure 2.2 to signify that we are interested in a subset of this generalization. As we discuss in the introduction, we are posing the hypothesis that proprioception can generalize better than exteroception to new environments. So in this framework, proprioception can be trained on human-labeled data from a small set of environments, and then exteroception can be automatically trained by the proprioception on a larger set of environments.

Brooks and Iagnemma [8] provide an example that fits into this two-step framework in figure 2.2. They implemented multiclass, self-supervised terrain classification for a planetary exploration rover. The property learned was one of three ground types: sand, beach grass, or rock. The proprioceptive features came from vibration measurements and the exteroceptive features came from camera data. In the first step, a human provided labels for one of the three ground types. These were used to allow the vibration features to build a multiclass classification model that could distinguish between these three ground types. In the second self-supervised step, the model built for the vibration features provided predictions on additional data, and this additional labeled data was used to train appearance features from the camera.

As discussed above, the reason why the first step is easier than the second step is because fewer labels are needed for the proprioceptive data than for the exteroceptive data. Brooks and Iagnemma argued this point. They showed how variations in lighting conditions caused the appearance of the ground type to change, whereas the vibration signatures remained consistent. In table V of their publication [8], they showed that predictions from the solely appearance-based models obtained much lower accuracies when trained on data from environments where the illumination was slightly different.

Angelova et al. [4] presented an instance of self-supervised learning that is quite different from the frameworks discussed so far. This framework is generalized in figure 2.3. In the first step, appearance features from a camera were modeled with unsupervised clustering into one of three ground types: soil, gravel and asphalt. These labels were then used in a second self-supervised step. In this second step, a new property was learned: wheel slip. In the second step, the teacher is a combination of two measurement types, proprioception and exteroception. The exteroception is the set of ground type predictions coming from the appearance features in the first step. The proprioception is a set of direct measurements of wheel slip; the difference in commanded velocities measured by wheel encoders and actual velocities measured by Visual Odometry (VO) gave a measure of pose discrepancy, which could tell the robot how much the wheels were slipping. The student was a set of slope estimates of the surrounding terrain, which is an exteroceptive modality separate from the appearance. A regression model was learned which, given a certain ground type, would turn the slope measurements into a wheel slip value.

Angelova’s framework in figure 2.3 is very different from the general two-step framework in figure 2.2. One difference is that the first step is unsupervised, so it requires no human labels. Another difference is that each of the two steps outputs a separate property set (first ground type, then wheel slip); in the general two-step framework, the same set of properties is estimated in both steps. In both frameworks, the labels from the first step

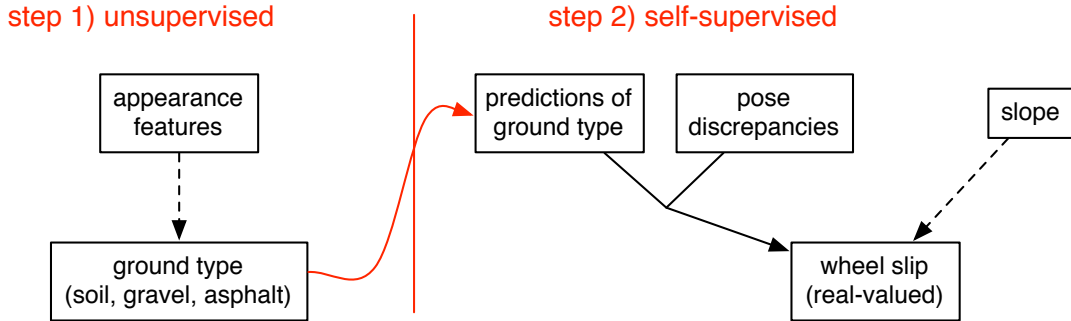


Figure 2.3: Example of an extension to the general frameworks, used in Angelova et al. [4]

are used as the teacher for the second step; but in Angelova’s framework, a set of direct measurements acts as another teacher in the second step.

Another large difference between these two frameworks is that appearance acts as a teacher in Angelova’s framework and as a student in Brooks and Iagnemma’s framework. Angelova’s approach was much more confident than Brooks and Iagnemma’s approach about the ability of the appearance model to generalize to new environments. They assumed that appearance features can be learned in an unsupervised fashion, and furthermore, the implicit assumption with unsupervised learning is that they would automatically generalize to new environments, since they did not need human labels at all. In later work, [3], [5], the authors relax the constraint that the appearance features must be learned in an unsupervised fashion. Instead of the two steps happening sequentially, they allowed them to happen simultaneously within an Expectation Maximization algorithm. In any event, the point of this work was not to characterize the variability of ground type from one environment to another, but rather to use ground type as input for learning a regression model that relates slip and slope. Nothing would prohibit this entire framework from being contained within a larger framework that used proprioceptive features to make the appearance models more robust to new environments.

More recent work extends the work from Brooks and Iagnemma, using again the framework in figure 2.2, but still only looking at ground terrain classes. Otsu et al. [26] distinguish between three ground classes: bedrock, soil and sand. The teacher is from a vibration signal and the student is from camera images. Hajjam and Rutherford [15] distinguish between five ground classes: grass, asphalt, dirt, woodchips and sidewalk. The teacher is a vibration signal from an IMU and the student is from camera images coming from an aerial vehicle.

## 2.3 Comparison of Approaches

Table 2.2 compares the self-supervised examples discussed in the previous section. The first four are examples of the direct one-step framework in figure 2.1, discussed in section 2.2.1, where the measurements from the teacher could directly be turned into values for the

property being estimated. The last two are the examples discussed in section 2.2.2, for the two-step frameworks depicted in figures 2.2 and 2.3, where the measurements from the teacher required some learning beforehand. The second column in the table makes this distinction: whether the measurements from the teacher were direct or learned. The double horizontal line after the first four rows highlights this distinction. Interestingly, this double line also highlights a fundamental difference in the third column: the *number* and *types* of properties that can be estimated by the learning model.

Authors	Direct or learned model for teacher	Properties estimated
Stavens, Thrun [33]	Direct	Terrain roughness
Kim et al. [19]	Direct	Terrain compliance
Howard et al. [17]	Direct	Terrain compliance
Ott, Ramos [27]	Direct	Terrain compliance
Wellington, Stentz [38]	Direct	Terrain compliance
Brooks, Iagnemma [8]	Learned	Sand, beach grass, rock
Otsu et al. [26]	Learned	Bedrock, soil, sand
Hajjam, Rutherford [15]	Learned	grass, asphalt, dirt, woodchips, sidewalk
Angelova et al. [4]	Learned, unsupervised	Wheel slip soil, gravel, asphalt

Table 2.2: Comparison of self-supervised learning implementations on mobile robots, where the teacher was proprioceptive. The double horizontal line after the first four rows divides the examples by whether the teacher had a direct analytical model or whether it first had to learn a model before going on to train the student.

First lets consider the *number* of properties estimated. For the examples that use the one-step framework (above the double line), the output was one quantity, such as terrain compliance or roughness. The quantity was binary for some examples and continuous for others, but only one quantity was being estimated. For the examples that use a two-step framework (below the double line), the output was a multiclass semantic label. In these cases, the semantic label distinguished between multiple ground types.

Now lets consider the *type* of properties estimated. For the first set of examples above the double line, the output was describing the physics of the vehicle’s interaction with the environment. For the second set of examples below the double line, the output was describing the material consistency of the environment, which is independent of the vehicle’s interaction with it. Remember that in all of these cases, proprioception is being used to understand the external environment by interacting with it. In the first set of examples, the direct physics of this interaction is being described; in this case, the motion of the vehicle. *Terrain compliance* is a measure of whether the robot is able to move through the object or not. *Terrain roughness* is a measure of whether the vehicle’s suspension bumps up and down as it drives over a certain ground type. These are both examples where analytical models (in this case equations of motion) can be used to turn the measurements into

property values. However, in the second set of examples, the authors measured material properties of the environment, such as whether the ground consists of sand, grass, rock, soil, gravel or asphalt. These properties are still being measured through interaction, but they are not a direct effect of this interaction; they exist independently of the interaction, but can still be measured through the interaction with *learned* models.

One can appreciate a trade-off between the two types of approaches. In the direct one-step framework, no data is needed to come up with the teacher’s model, so a minimal amount of *human effort* is required. However, the *semantic richness* of the properties estimated is limited by what can be modeled analytically. In the learned two-step framework, some data is needed, and sometimes this data needs human labels as well, so more *human effort* is required. However the *semantic richness* of what can be estimated is greater.

We now compare all of the learning approaches discussed so far for terrain classification on mobile robots where proprioception is used. We use this as a basis for moving forward with the work addressed in this thesis. This includes a comparison of the supervised approaches from section 2.1, and the self-supervised approaches from section 2.2. Table 2.3 gives this full comparison. Each row is a group of authors. Each column is one of the problem dimensions that we define in section 1.2.

The first dimension is the *semantic richness* of the world model. This is a measure of the *number* and *types* of properties that can be estimated. For the *number* of properties estimated, the value is either “single” or “multiple” in the table, referring to whether only one quantity can be estimated, or if a multiclass model can be built. For the *types* of properties estimated, this can be either a direct measurement of the motion (in which case we write “motion”) or it can be a measurement of the material properties of the environment (in which case we write “material”). Another distinction we make about the *type* of property is whether the terrains are limited to ground types or if the model can handle above-ground types. (This is denoted as “ground” or “above-ground”.)

The second dimension is the amount of *human effort* required to build the model. This can have the values of low, medium or high. “Low” human effort means that no human labels are needed to transform the proprioceptive measurements into the properties being estimated. This corresponds to the examples of the one-step framework, where the measurements can automatically be turned into property values. “Medium” human effort means that human labels are needed for a small set of locations, but the assumption is that the learned model will be able to generalize well to new locations. This corresponds to the general two-step framework in figure 2.2, used by Brooks and Iagnemma [8], Otsu et al. [26], and Hajjam and Rutherford [15]. “High” human effort means that new data would be needed for a large number of locations because the features do not generalize well. Note that none of the examples listed have a value of high, because they were all using proprioceptive data, which allows them to generalize well to new environments. (Note that Angelova et al. [4] does not quite fit this assumption, but the details of this unique instance were already discussed in section 2.2.2. Also, since Angelova et al. uses unsupervised learning for the first step, empirical data is needed but the data does not require human labels, so we denote the effort as low/medium.)

The third dimension is the amount of *robot effort* that is required. This directly corresponds to whether the example was part of a self-supervised scheme or not (whether

it came from section 2.1 or section 2.2). If it was solely a supervised scheme, then the learning was only using proprioceptive data. If it was a self-supervised scheme, then the proprioception was used to teach the exteroception. In the former case, since no exteroception is used, the robot must physically interact with every patch of terrain that it wants to estimate. Therefore, the amount of robot effort required to map an entire region is “high.” In the latter case, the proprioceptive estimates can be mapped onto a larger region by the exteroceptive estimates, and so only some of the terrain needs to be traversed. Therefore, the amount of robot effort is “medium.” Note that none of these examples have a value of “low” because proprioception is always being used, and therefore some amount of terrain needs to be traversed to collect the proprioceptive data.

The table shows that for the first dimension (semantic richness), only rows two through four are estimating properties “above-ground.” These examples fall into the “single” and “motion” categories because they are all examples from the one-step framework, where motion measurements of the interaction are analytically being modeled as a single property value. Rows six through ten fall into the “multiple” and “material” categories but are all limited to properties of the “ground.” For instance, consider the example of distinguishing between a rock and a bush that we discuss in the introduction. This is a classification of the *material* property of an object *above-ground*. None of the examples listed in this table attempt to estimate this type of property.

The last row of the table denotes how the work in this thesis extends the results from prior work. The first column states that we increase the level of semantic richness provided by the prior work by estimating multiple, above-ground, material properties of the terrain. The second and third column state that we require medium amounts of both human and robot effort, which means that we keep our effort levels comparable to prior approaches. Section 2.4 describes the thesis problem explicitly.

<b>Authors</b>	<b>1) Semantic richness</b>	<b>2) Human effort</b>	<b>3) Robot effort</b>
Stavens, Thrun [33]	Single, motion, ground	Low	Medium
Kim et al. [19]	Single, motion, above-ground	Low	Medium
Howard et al. [17]	Single, motion, above-ground	Low	medium
Ott, Ramos [27]	Single, motion, above-ground	Low	medium
Wellington, Stentz [38]	Single, motion, above-ground	Low	medium
Stavens et al. [34]	Single, motion, ground	Medium	High
Brooks, Iagnemma [8]	Multiple, material, ground	Medium	Medium
Otsu et al. [26]	Multiple, material, ground	Medium	Medium
Hajjam, Rutherford [15]	Multiple, material, ground	Medium	Medium
Ojeda et al. [25]	Multiple, material, ground	Medium	High
Weiss et al. [37]	Multiple, material, ground	Medium	High
Coyle, Collins [11]	Multiple, material, ground	Medium	High
Angelova et al. [4]	Multiple, material, ground	Low/medium	Medium
Christie, Kottege [10]	Multiple, material, ground	Medium	High
Zhao et al. [42]	Multiple, material, ground	Medium	High
Yu, Lee [41]	Multiple, material, ground	Medium	High
Valada et al. [35]	Multiple, material, ground	Medium	High
<b>Thesis Problem</b>	<b>Multiple, material, ground and above-ground</b>	<b>Medium</b>	<b>Medium</b>

Table 2.3: Comparison of proprioceptive learning approaches on mobile robots, and how they address the three dimensions of the problem space



## 2.4 Thesis Problem

In the last row of table 2.3, we identified the problem that we address in this thesis. That section directly revisited the three problem dimensions listed in section 1.2, and specifically addressed how we extend prior work to address these dimensions. Below, we flesh out the three problem dimensions again, and how we address them in this thesis.

1. We increase the *semantic richness* of our world model. Specifically, we estimate material properties of the environment, both of the ground and above the ground. The terrain classes we estimate include:
  - (a) Ground types: grass, grass and leaves, pavement
  - (b) Above-ground types: soft vegetation, bramble, bushes, trees
2. We decrease the level of *human effort* required. This is the number of human-labeled data points needed for training the model. It is denoted as a *medium* amount in table 2.3 because we recognize that we need some human-labeled data so that a more complex model can be learned and in turn so that more types of properties can be estimated. (Note that from the previous authors, the only ones that require a *low* amount of human effort use analytical models, which puts a strong limitation on what types of properties can be estimated.) Our goal is to decrease the number of data collection locations that require human-labeled data. This is achieved by using a proprioceptive model. Our hypothesis is that a proprioceptive model will generalize to new locations better than an exteroceptive model. By using a proprioceptive model, we can therefore decrease the number of locations for which human-labeled training data is needed.
3. We decrease the level of *robot effort* required. This is the amount of terrain that the robot must interact with in order to estimate properties about the surrounding environment. It is denoted as *medium* in table 2.3, which means that the robot has to interact with some of the terrain but not all of it. Our goal is to decrease the fraction of terrain within each test location with which the robot must interact. This is achieved through self-supervision. By using the proprioceptive model to train the exteroceptive model at each new location, we leverage the ability of exteroception to collect data without interacting with the terrain. In this way, we can bootstrap the proprioceptive predictions onto a larger terrain map.

## 2.5 Thesis Outline

Now that we have defined the thesis problem in relation to prior work, we present an outline for the rest of this document.

Chapter 3 describes our experimental setup. We describe the two robotic platforms used: the Gator in our early work, and the retrofitted LAGR in our more recent work. We describe the sensors used for the proprioceptive and exteroceptive suites. We then describe the data collection procedures.

Chapter 4 describes how we implement the full self-supervised system. We describe

how we registered the proprioceptive and exteroceptive data. We introduce notation to help us keep track of the different subsets of data that we used for the different components of the system. We delineate the procedure for the four main experiments that we use in order to test and evaluate the system. We conclude the chapter by revisiting the thesis problem in section 4.4, describing how these experiments will provide evidence that we have addressed the problem.

Chapter 5 examines the proprioception module. We start by surveying literature in acoustics and vibration, which provides a basis for the proprioceptive feature extraction and learning techniques that we use. We then describe the technical approach to these feature extraction and learning techniques. We present supervised proprioceptive experiments on our early Gator platform. We then extend this work to experiments on the LAGR platform, with a more sophisticated proprioceptive sensor suite. Part of using this more extensive suite involves a sensor selection process, which we describe.

Chapter 6 presents the exteroception module. We describe the technical approach to our feature extraction and learning techniques. The feature extraction involves an unsupervised feature learning step, and we benchmark this against more traditional features. We then present experiments on the LAGR platform, showing results on supervised exteroception, comparing the different techniques. This chapter lays a foundation for incorporating the supervised exteroception into the self-supervised system.

Chapter 7 presents the experiments and results for the full self-supervised system. The framework we present in chapter 4 delineates the four main experiments that will be run, and this chapter runs these experiments and presents results. We conclude the chapter in section 7.5 by discussing how the results provide evidence to the claims in our thesis problem.

# Chapter 3

## Experimental Setup

In this chapter we discuss the robotic platforms used and the data that we collected on these platforms. We discuss each vehicle we used, and what sensors we mounted on those vehicles. This includes our early work on the Gator platform, as well as our more recent work on the LAGR platform. We also discuss the data that we collected on each vehicle. In chapters 5, 6 and 7, we use this data in various modules of the algorithm. For the LAGR platform, we explain the data collection procedure in detail in section 3.4. This acts as a precursor to introducing the full self-supervised system in chapter 4.

### 3.1 Gator Platform and Data Collection

We conducted our early work on the Gator platform, depicted in figure 3.1. This is a robot built from a John Deere E-Gator vehicle. We collected data from microphones and a vibration sensor mounted on the vehicle. We manually drove the vehicle through various outdoor environments. Two Blue Snowball microphones recorded sound, one mounted to the front grill and one on the back bed. We also mounted a webcam to the front grill, which we used for hand-labeling assistance. We mounted a Measurement Specialties IEPE accelerometer near one of the front wheel axles. It sensed vibrations between the wheel and the ground and was specifically mounted close to the source of the interaction, before the vehicle suspension. (See section 3.2 for more details on these sensors.) A National Instruments data acquisition board performed the analog-to-digital conversion of the IEPE device. A laptop running Windows interfaced with this board. Another laptop running Ubuntu interfaced with the microphones as well as the main embedded computing architecture on the robot.

We collected data for six outdoor terrain classes by having the vehicle interact with each terrain. The classes fall into two main categories: benign and hazardous terrain interactions. We have three classes for each category, as listed below:

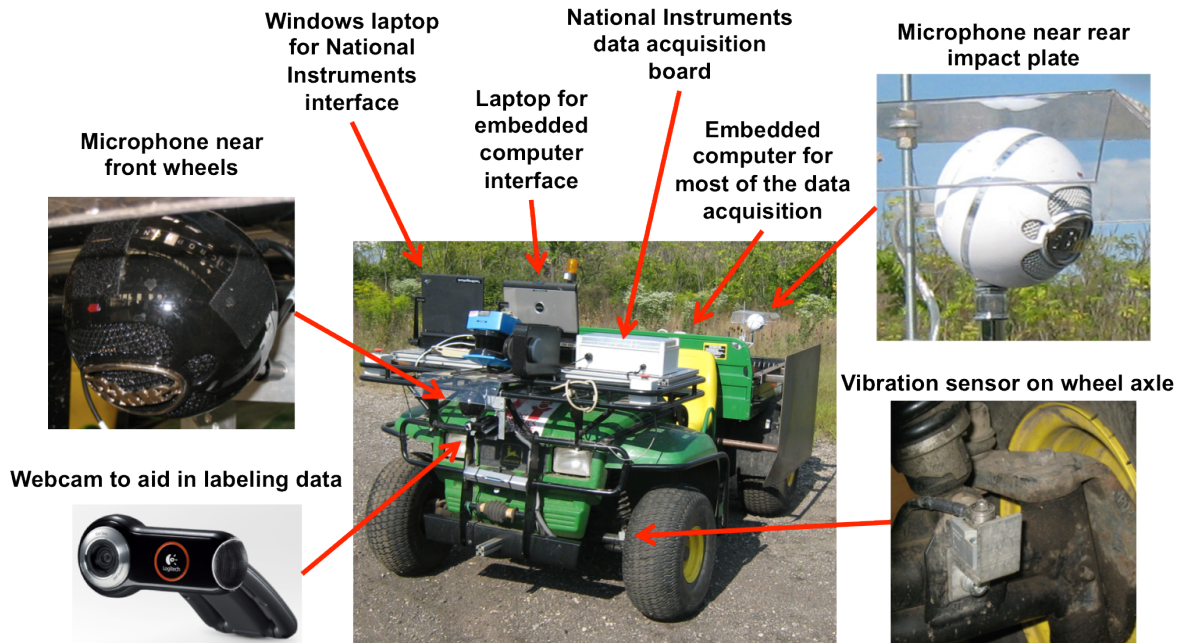


Figure 3.1: The E-Gator experimental platform, highlighting the sensors and computing infrastructure. Close-ups are shown of the two Snowball microphones, the IEPE vibration sensor, and the webcam.

1. Benign terrain interactions:
  - (a) Driving over grass
  - (b) Driving over pavement
  - (c) Driving over gravel road
2. Hazardous terrain interactions:
  - (a) Splashing in water
  - (b) Hitting hard objects
  - (c) Wheels spinning freely when stuck in slippery terrain

For each class, we collected data from multiple locations. We chose locations with differing characteristics so that there was variation within each class. This prevented our classifiers from overfitting to specific data sets and proved the ability of these models to generalize to new environments.

For the benign terrains, the sounds differed according to how the tires interacted with the road surface. For the *grass* terrain, the locations varied from clean-cut lawns to un-maintained, park-like settings. For the *pavement* terrain, the locations varied from new asphalt parking lots to older concrete roads. For the *gravel* terrain, the locations varied from dirt roads with a few slag pebbles to dense collections of crushed limestone.

For the *splashing in water* terrain, we drove into puddles and shallow streams to collect the sound of splashing produced by the tires and undercarriage of the vehicle. We collected

data from three puddles formed from rain in ditches along dirt roads (figure 3.2(a)), where the dirt roads had varying levels of mud and ditch depth. We also collected data from two locations in a naturally running stream of water about one foot deep (figure 3.2(b)), where each location had varying levels of terrain roughness and water flow.



(a) Puddle of water

(b) Stream with running water

Figure 3.2: Examples of various environments used for the *splashing in water* class, speaking to the ability of the classifiers we train to generalize across varying environments within each class.

For the *hitting hard objects* terrain, we drove the side of the vehicle into large rocks and other hard objects (figure 3.3). To prevent damage to the vehicle, we rigidly attached a large steel sheet to the side of the vehicle, similar in thickness and material to the vehicle frame. The collisions always happened against this steel sheet. We collected data from six different objects, hitting each object multiple times from multiple angles. Our data set contains 20 to 30 collisions with each object, consisting of banging and scraping sounds as the vehicle hits the object and continues to try to drive past it. We used objects of various shapes and sizes. One was a rectangular cement block, two were sandstone rocks, two were rocks formed from a concrete gravel mix, and one was a rock formed from molten slag.

For the *wheels spinning* class, we collected data when the vehicle got stuck in slippery terrain such as mud or snow, or from uneven terrain such as a ditch. These events consisted of the tires spinning freely in place with no traction. This data was the result of our vehicle actually getting stuck when trying to collect data for other classes. We captured this data type in over ten different locations, with each location consisting of multiple traction loss events. These locations included ditches in off-road meadows, ditches in a snow-covered trail, uneven terrain in the stream mentioned above, and various ditches next to the rocks mentioned above.

We positioned the back microphone near the mounted steel sheet so that we could capture the sounds of the vehicle hitting hard objects on this sheet. We mounted the other microphone and the vibration sensor near the front wheels, and all other vehicle-terrain interactions happened in this region. Ideally, we would have had microphones and

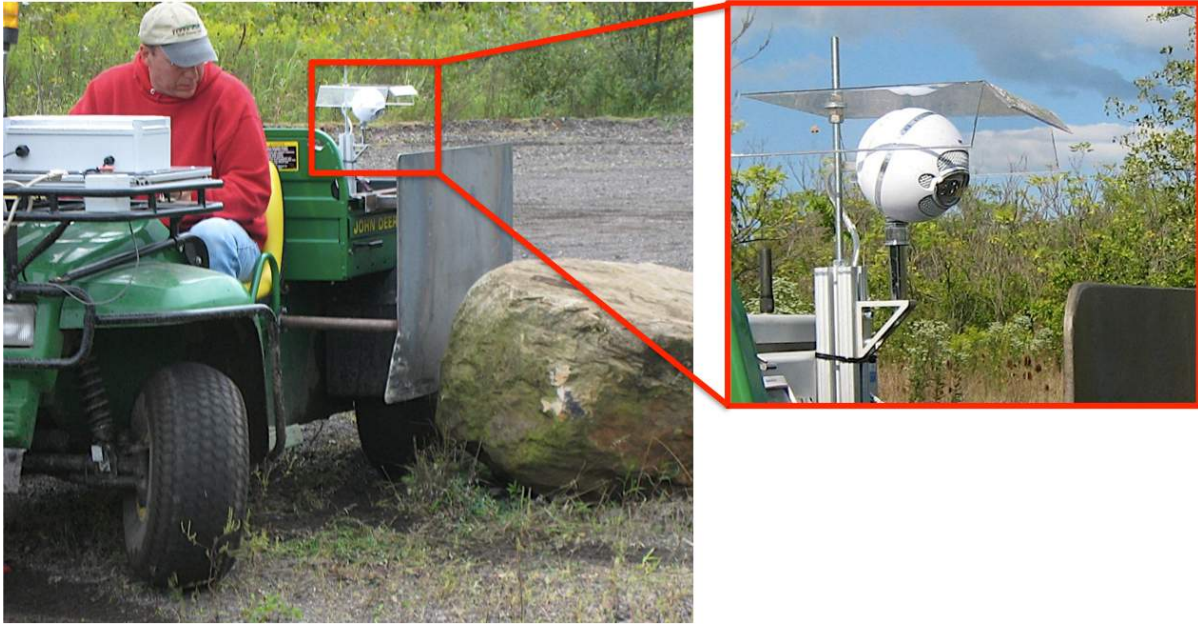


Figure 3.3: Example of a rock used for the *hitting hard objects* class. *Left*: The vehicle driving into a rock. The location of the collision is against the steel metal plate mounted to the side of the vehicle, to protect the vehicle’s frame from damage. *Right*: A closeup of the microphone mounted on the back of the vehicle near to the collision location.

vibration sensors all over the vehicle, but cost and experimental setup times were limiting resources. For all of the trials, we maintained a roughly constant speed between 2 m/s and 3 m/s. Earlier experiments showed that speed can have a large effect on the sound of the interaction, so we controlled this variable here.

To label the data, the starting and ending timestamps of each interaction event had to be determined. For example, a particular sound file might contain two events: driving through a puddle of water and driving along the road next to the puddle. Timestamps must be determined in order to break such a sound file up into separate sequences for these two events.

We developed interactive software to aid in labeling our data, which incorporates a combination of webcam images, time series plots, and audio feedback. We mounted a webcam on the vehicle near one of the microphones, providing a video stream of the environments that the vehicle was encountering (refer to figure 3.3). Hand-labeling involved listening to the sound files and using time series plots of this data to visually zoom in on interesting sequences and graphically hand-select the starting and ending timestamps of each event. Section 5.2 discusses the hand-labeling in more detail.

After labeling was complete, we had sequences of time series data where each sequence represented an event of interacting with a specific class. We split these sequences up into short time windows. Each window is then a data point which is subsequently used for feature extraction and classification. Each window generates a terrain class prediction. Chapter 5 discusses the pipeline for turning a data point window into a class prediction. We

empirically chose a window size of 200 ms, with 50% overlap between successive windows. Refer to section 5.2 for choice of window size.

Table 3.1 shows the number of data points we collected with the Gator for each terrain class. Note that there is much less data for the last two classes because these interactions involved short events, whereas for the first four classes we had more continuous stretches of terrain with which to interact.

Class	Number of data points
grass	2737
pavement	3233
gravel	8333
water	5171
hard objects	268
losing traction	333

Table 3.1: Number of Data Points for Each Class on Gator Platform

## 3.2 Proprioceptive Sensing Modalities

Our work on the Gator platform provided evidence that proprioceptive sensing generally and acoustics specifically could classify terrain with invariance. (This is discussed in section 5.5.) This led us to further explore these sensing modalities. In transitioning from the Gator to the LAGR platform, we greatly enhanced our proprioceptive sensing suite. This section describes how we chose the sensors to make this enhancement. First, we analyze the prior work examples already given and discuss how the sensing modalities used there played a factor in the amount of semantic richness that could be estimated. Then, we look at how we extended this work with similar sensing modalities that have fewer limitations. The section ends with a discussion of the sound and vibration modalities we used.

Table 3.2 lists the same authors that we discuss in chapter 2, but compares the first dimension of semantic richness to the proprioceptive sensing modalities used. The apparent trend is that the measurements used to estimate direct motion properties were providing low dimensional features, while the measurements used to estimate material properties about the environment were providing high dimensional features. All of these sensing modalities output some form of time series data. If a data point is considered to be a small window of time, then the dimensionality of the data point is a measure of how much information is contained within that window.

First lets consider the low-dimensional sensing modalities. Howard et al. [17] used bumpers, and the measurements from the bumpers were binary values: whether the bumper had hit something or not. This is a one-dimensional value with a discrete binary scale. Kim et al.[19] used bumpers, IMU, and motor current. This approach used slightly more data, but they purposely allowed this data to redundantly overlap, reducing it to the same one-dimensional binary value: whether the robot had hit something or not. Stavens and Thrun [33] used shock measurements provided by the signal from the z-axis of the accelerometer in their IMU. After removing the gravity bias, they integrated the z-axis

Authors	Semantic richness	Measurements used
Stavens, Thrun [33]	Single, motion, ground	Shock
Kim et al. [19]	Single, motion, above-ground	Bumpers, IMU, motor current
Howard et al. [17]	Single, motion, above-ground	Bumpers
Wellington, Stentz [38]	Single, motion, above-ground	Filtered pose
Stavens et al. [34]	Single, motion, ground	Shock
Brooks, Iagnemma [8]	Multiple, material, ground	Vibration
Ojeda et al. [25]	Multiple, material, ground	Vibration
Weiss et al. [37]	Multiple, material, ground	Vibration
Coyle, Collins [11]	Multiple, material, ground	Vibration
Angelova et al. [4]	Multiple, material, ground	Appearance, pose discrepancies

Table 3.2: Comparison of the measurements used for the inputs and the semantic richness that is output from proprioceptive learning approaches on mobile robots

amplitude for each short time window. This integral is a measure of the energy along the z-axis, which is proportional to the bumpiness of the ground. This is a one-dimensional real value: the amount of energy in the window. Wellington et al. [38] used a 6-DOF pose estimate, so they had six dimensions to work with. They used a variety of sensors that were fed into an Extended Kalman Filter (EKF) to output this estimate. In all of these examples, the measurements of interest slowly varied over time. So for some small window size, the data point can be thought to have a small number of values (either a single value or six values in the case of the 6-DOF pose estimate).

Now lets consider the high dimensional measurements where mostly vibration data was used. Vibration data can be obtained from the z-axis of an accelerometer, similar to the shock measured in Stavens and Thrun [33], but the signal is processed differently. Instead of integrating each small window to obtain a single energy value, the Discrete Fourier Transform (DFT) of the window is taken to obtain an entire distribution of values. For a 0.25 second window sampled at 800 Hz, there would be 200 samples, and 100 bins, resulting in a 100-dimensional vector.

Note that any time series signal can be sampled at any rate, depending on what type of analog-to-digital (A/D) conversion the system is using. The higher the sampling rate, the more frequency bins there are in the DFT distribution. But if the natural environment being measured does not produce high frequencies, then using the higher frequency bins is futile. An example of this might be a temperature measurement that slowly varies over time. There are no high frequency components in this signal, so it is inherently low-dimensional. In the 6-DOF pose estimate from Wellington and Stentz’s work, each of these six dimensions is similar to temperature: it is an amplitude that varies slowly over time. So there is no way to represent this data significantly in a high dimensional space.



<b>Authors</b>	<b>Vibration sensor</b>
Brooks, Iagnemma [8]	Contact microphone
Ojeda et al. [25]	Accelerometers, gyroscopes from an IMU
Weiss et al. [37]	Accelerometers from an AHRS
Coyle, Collins [11]	Various measurements from an IMU

Table 3.3: Comparison of sensors used for measuring vibration

Exteroceptive measurements such as appearance from cameras are also capable of estimating complex properties, as is demonstrated in Angelova et al. [4]. In general, we will use exteroceptive data as the student and not the teacher in a self-supervised framework, but the student must still be able to estimate the same properties, just not as reliably. So it is still important to discuss how exteroceptive sensing falls into this dimensionality argument. Appearance data is also time series data because it is a sequence of images over time. We can think of each pixel as having four amplitudes (red, green, blue and intensity) that vary with time. Each of these amplitudes varies slowly over time, similar to temperature, so each pixel is low-dimensional. However, features for image data are calculated for some spatial window around each pixel. Each window has many pixels, so for each moment (or small window) in time, we have many dimensions spatially. Range data from stereo cameras or ladar is also high dimensional for similar reasons. Instead of pixels, we now have 3d voxels, and for each voxel we have one dimension for range. Looking at cubic windows around each voxel, we again have many dimensions spatially.

So far, we have noted the trend that measurements with many (non-redundant) dimensions provide more information, which allows for more semantic properties to be estimated given the appropriate model. All of the proprioceptive examples that used high dimensional measurements are using *vibration* data. (The one exception is the exteroceptive appearance data used in Angelova’s work.) One of our research goals was how to extend this prior work in vibration sensing with new sensing modalities. To answer this, we look more specifically at the sensors used to collect these vibration measurements. This sensor-to-measurement comparison is depicted in table 3.3.

With the exception of the contact microphone in Brooks and Iagnemma’s work [8], the rest of the examples used measurements from an IMU or an AHRS, both of which measure pose and velocity. These systems usually have low-pass filters on the analog signal before the A/D conversion to remove high frequency components. (The cut-off point is around 25 Hz for certain IMU’s.) When measuring pose or velocity, these high-frequency components are considered to be unwanted noise. However, for the alternate goal of measuring vibration, this higher bandwidth could be very useful, and so the low-pass filters could be removing useful information. With that being said, even with a 25 Hz cutoff point, a signal that is sampled at 50 Hz (Nyquist sampling rate) with a one second window would still result in a 25-dimensional feature vector, which is still relatively high compared to one-dimensional direct measurements.

These authors also could have been sampling the signal before the low-pass filters if the IMU or AHRS system allowed for this level of interface control. The cut-off point

for meaningful information is then determined by the physics of the sensing element or the vehicle-terrain interaction. The sensing element will have a natural limit to what bandwidth of frequencies it can sense. For an accelerometer, this is usually related to the resonance of the spring-mass system inside the device. Cheap accelerometers usually have an upper limit of 500 to 1000 Hz. The accelerometers used within IMU's tend to be of this variety since they will be attached to a low-pass filter anyway.

Some accelerometers are tailored for measuring higher bandwidths (up to 10 kHz). However, the relevant question is whether the natural physics of the vehicle-terrain interaction contains meaningful frequencies in this upper range, or if this range is just noise. If most of the information is in the lower part of the spectrum, then the accelerometers passed through low-pass filters inside of an IMU might be sufficient, and such a device can serve the dual purpose of measuring pose and vibration signatures at the same time. However, if the higher bandwidths contain useful information, then low-pass filters should be avoided, and the sensing element itself should perhaps be tailored to focus on these higher bandwidths. We compared these different vibration modalities in our work. Specifically, we compared:

- Accelerometers from within an IMU that potentially have low-pass filtering
- Accelerometers similar to those used within IMU's but separated from this larger system, which eliminates low-pass filtering
- Accelerometers that are tailored to pick up higher bandwidths (up to 10 kHz)

*Sound* is another proprioceptive sensing modality that we explored, overlapping in some ways with our exploration into vibration sensing. As humans we have empirical proof that the sounds we hear help us understand a better model of the world, so it makes sense to explore this modality from a bioinspired perspective. For instance, the reader might imagine stepping into a puddle of water. If he was not using his eyes to look down so that he did not *see* the interaction, and he had waterproof boots on so that he did not *feel* the interaction, then he would still be aware of the event because he would *hear* it, and his brain would understand how to make sense of what he heard. So we know that our brains are capable of learning models from the sounds we hear, which help us distinguish between different interactions. This experiential evidence suggests that machines can also learn models from sound signals. (This is similar to why computer vision has been such a popular sensing modality for machine learning. Researchers had experiential evidence that visible light detected by their eyes could be used to learn mental models, so they expected that machines could learn these models as well.)

Sound and vibration are overlapping terms. Sound is really a specific type of vibration. Vibration is a mechanical oscillation of pressure that can travel through a solid, liquid or gas. Sound is more specifically vibration within the human hearing range (20 to 20,000 Hz). The vibrations explored in prior work, as compared in table 3.3, are more specifically traveling through a solid; in this case, the physical body of the robotic vehicle. Sound traveling through air can be measured by using a microphone of some variety, whereas vibrations traveling through solids can be measured by various sensors as outlined in table 3.3. Note from this table that Brooks and Iagnemma [8] use a contact microphone, which measures

vibrations through solids, as opposed to air microphones used for measuring sound through air.

It follows naturally to view sound measured through air as an extension to prior work in vibration measured through solids. Furthermore, air microphones tend to have higher frequency ranges than sensors that measure vibration through solids, since these waves travel at higher frequencies through air than through various solid materials. This further motivates the use of sound from the dimensionality argument made above.

Another motivation for measuring sound traveling through air (as opposed to solids) is that we can more successfully measure the interaction of the vehicle with non-rigid bodies, such as vegetation or water. When the vehicle interacts with these types of objects, the objects will be compressed, making less of an impact on the vehicle. If there was a hard impact, the energy from this impact would be partly transferred to the vehicle as vibrations through the solid. But if the body is compressible, the vibrations from this compression would be mostly transmitted to the surrounding air.

We summarize in the following list our motivations for experimenting with vibration and sound sensors:

- Motivations for measuring *vibration traveling through solids*
  - Prior work has shown success in using vibration as a higher-dimensional, proprioceptive measurement for estimating a larger variety of environmental properties.
  - The limitations on the frequency ranges of different vibration sensors suggest that further experiments be conducted to determine which sensors yield the most information.
- Motivations for measuring *sound traveling through air*
  - As humans, we have experiential evidence that terrain interactions can be distinguished from sound signals traveling through air.
  - It is a particular type of vibration, so in the spirit of exploring different vibration sensors, sound should also be explored.
  - It exhibits higher frequency bandwidths than vibration through solids, so it could potentially give us more information.
  - It has more potential to measure interaction with compressible objects, such as water and vegetation.

Table 3.4 lists microphones and vibration sensors with which we have experimented. The Blue Snowball is a relatively low-cost microphone that we also used in our early Gator work. This is an omnidirectional condenser microphone designed for indoor studio recordings. This means the sound quality is good, but its ability to withstand harsh outdoor environments is not known. It comes prepackaged with an A/D conversion and a USB output.

The Voice Technologies VT500Water is an omnidirectional electret condenser microphone. It is a miniature lavalier microphone designed for pinning onto a shirt collar during broadcasting events. This model is waterproof and can be submerged completely without

Sensor	Signal Medium	Company	Model	Bandwidth (Hz)	Price
Capacitor condenser microphone	Air	Blue Microphones	Snowball	10,000	\$100
Electret waterproof microphone	Air	Voice Technologies	VT500Water	20,000	\$300
accelerometer inside the IMU	Solid	Xsens Technologies BV	MTi	100	\$1500
Typical cheap accelerometer found in IMU's	Solid	Analog Devices	ADXL 335	500	\$30
High-end piezoelectric vibration sensor	Solid	Measurement Specialties	7100A	10,000	\$500
Guitar pickup, contact microphone	Solid	Signal Flex	SF-20	10,000	\$20

Table 3.4: A list of the sound and vibration sensors that we used in our experiments.

any damage. Since it is small and waterproof, it is ideal for mounting near the wheel axles of a robotic vehicle, where it can fit into a small space while being exposed to many environmental factors. The device has an analog XLR output. This is a professional balanced audio output. We handled the A/D conversion and power supply with an XLR to USB converter. The A/D device used was the Shure X2u.

The Xsens MTi is the IMU already mounted on our robotic platform, discussed in section 3.3. It is a miniature, gyroenhanced Attitude and Heading Reference System (AHRS). It provides 3D orientation, 3D rate of turn, 3D earth-magnetic field data, and 3D calibrated acceleration. The 3D acceleration is the only signal we use, and we treated this as three separate x,y, and z vibration signals. For the z signal, we shifted out the gravity bias.

The Analog Devices ADXL 335 is a low-cost, low-power accelerometer. The ADXL line contains examples of accelerometers that one might find within an IMU, and in general they are ubiquitous in mobile robotics applications. As discussed above, we purposely tested these accelerometers separately from IMU's to avoid any low-pass filtering, and then compared them to the ones contained within IMU's. We used these sensors with a

prepackaged breakout board from Adafruit Industries for the necessary signal conditioning, and then used a LabJack data acquisition board for the A/D conversion.

Measurement Specialties develops a line of accelerometers that are much more expensive but provide a much higher bandwidth. The 7100A model that we used is an example of an Integrated Electronics Piezoelectric (IEPE) accelerometer, which has prepackaged signal conditioning to provide an amplified signal with less noise. This sensor is also completely sealed and waterproof. Because of the special conditioning, the only reliable way to interface with it is to use an A/D board that is tailored to sound and vibration measurements. We used a National Instruments 9234 data acquisition board for this purpose. To interface with the drivers for this board, we used a Windows Virtual Machine running on one of the vehicle’s embedded computers. The Linux drivers for this board were not developed at the time of this experimental setup.

The Signal Flex SF-20 contact microphone is the same model used in Brooks and Iagnemma’s work [8]. A contact microphone is also known as a guitar pickup, since it is predominantly used for this purpose, suctioned onto the side of an acoustic guitar. Its sensing element is a piezoelectric ceramic transducer. These are very cheap devices which are not at all calibrated, but are interesting still because they pick up high bandwidths as compared to vibration signals coming from accelerometers.

Time was spent characterizing each sensor to make sure that the capture levels and voltage ranges were appropriate so that the signals did not clip. We mounted all of these sound and vibration sensors on the LAGR platform, as discussed in the following section.

### 3.3 LAGR Platform

Figure 3.4 shows the experimental platform on which we conducted experiments. We built this from the original Learning Applied to Ground Robotics (LAGR) platform, which was used across multiple institutions in previous work. The module from the original platform that was responsible for motion control, localization, and remote control interfacing has been maintained. This module includes sensors such as the wheel encoders and the IMU, as well as the computing infrastructure for the low-level vehicle control. The rest of the computing infrastructure has been upgraded, and new sensors have been mounted and time synchronized.

Figure 3.5 highlights the new exteroceptive sensors that have been mounted. This includes a pair of high definition range (HDR) Pixim cameras and a Wobbler. The pair of cameras are forward-facing, with one angled slightly to the left and one angled slightly to the right, for a wider field of view. They are enclosed within a protective waterproof housing unit. The Wobbler is a SICK LMS 151 unit contained within a mechanism that allows for roll and pitch motion. Having two axes of motion renders a 3d point cloud with more uniform density. The mechanism for controlling the roll-pitch motion is a unique device developed by staff at the National Robotics Engineering Center (NREC). The cameras were intrinsically calibrated, and the cameras and lidar were extrinsically calibrated to each other, rendering colorized 3d point clouds. The calibration process involves moving a checkerboard poster around the field of view of the cameras and the Wobbler, as the robot



Figure 3.4: LAGR Platform

is standing still.

Figures 3.6 and 3.7 depict the proprioceptive sensors discussed in section 3.2. Starting with figure 3.6, the top left closeup is the Blue Snowball condenser air microphone. We have this mounted high on the vehicle since it is large and not waterproof. This is the same microphone we used in our early work on the Gator platform, discussed in section 3.1. The middle left closeup is the IMU, which was part of the original LAGR platform. This is located in the vehicle mast. We use the xyz acceleration signals coming from the IMU as vibration signals. The bottom left closeup is the VT500Water Electret air microphone. Since it is small, it can be tucked away in small places on the vehicle. Since it is waterproof, it can be low on the vehicle, because we don't have to worry about it getting splashed with water or mud. The one shown in the red box is above the bumper, and there is another one near the vehicle's axle.

The closeups on the top and bottom right of figure 3.6 show the suites of vibration

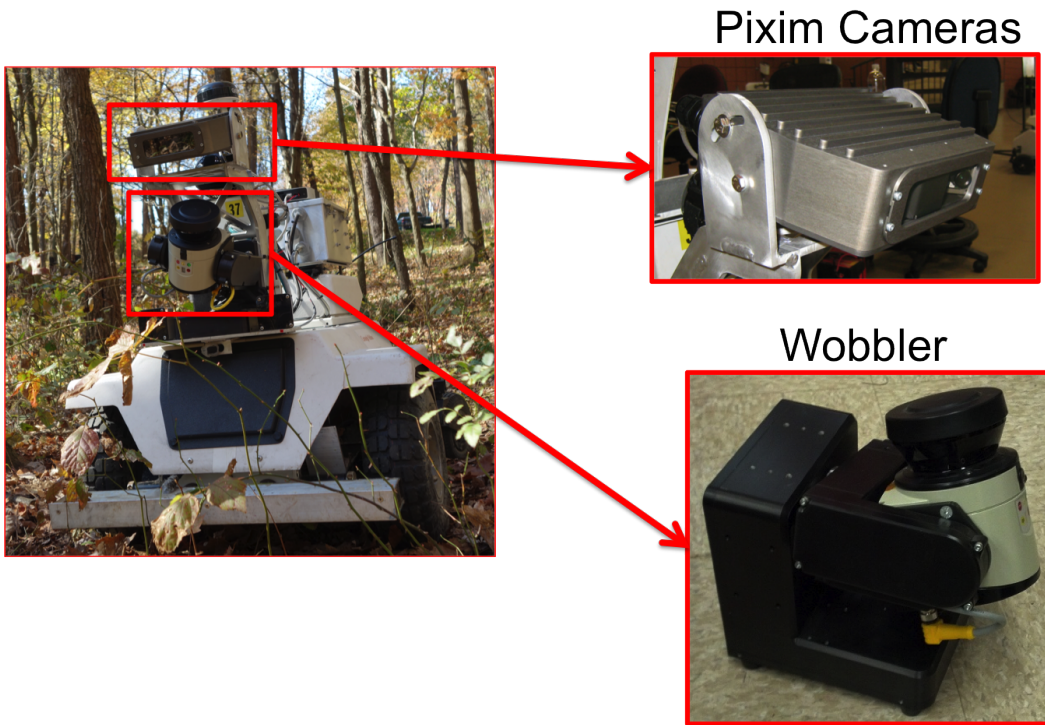


Figure 3.5: Exteroceptive Sensors on LAGR Platform

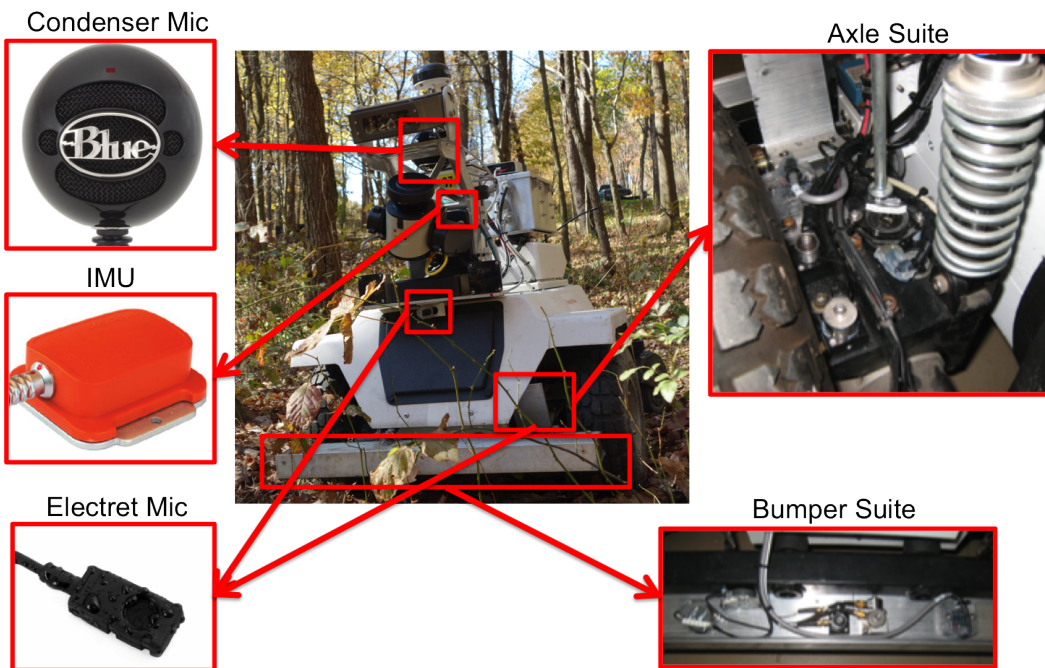


Figure 3.6: Proprioceptive Sensors on LAGR Platform, High Level

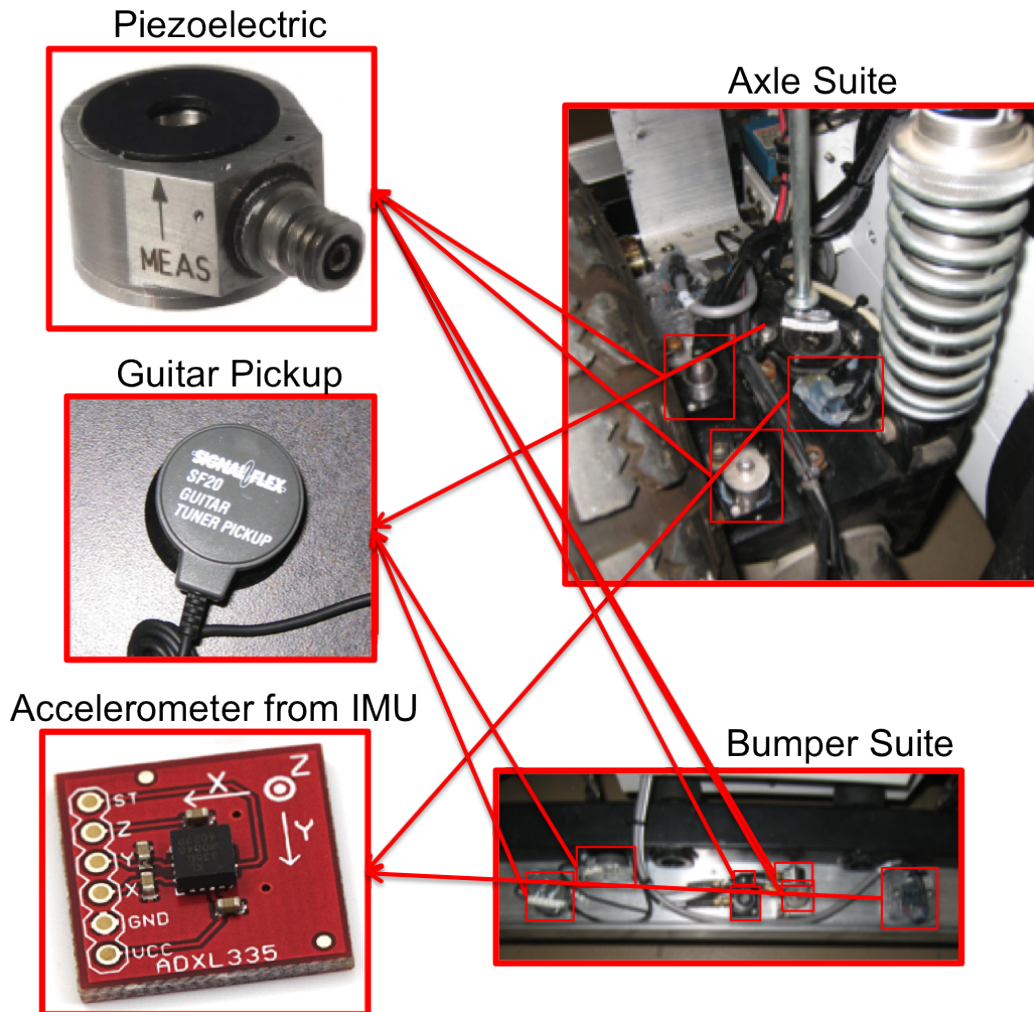


Figure 3.7: Proprioceptive Sensors on LAGR Platform, Inside the Axle and Bumper Suites

sensors mounted on the axle and the bumper of the vehicle, respectively. Note that on the center figure of the vehicle itself, these axle and bumper suites are enclosed by metal shields to protect them. In the closeups, we have removed the shields so that the sensors inside can be seen. The axle will predominantly pick up wheel-terrain interactions, and the bumper will predominantly pick up above-ground terrain interactions when the vehicle drives into something. Figure 3.7 shows closeups of the sensors in these suites. The top left closeup is the Measurement Specialties Piezoelectric vibration sensor. This is a high-end sensor that allows for conditioned signals up to 10kHz. It measures vibration along one axis. It comes in 10g and 50g varieties. On the axle, we have each variety mounted in the upward direction. On the bumper we have each variety mounted in the forward and upward direction. They are mounted on triaxial mounting blocks that come as an accessory to these sensors. So we have six of these sensors total:



- one 10g sensor on the axle in the upward axis
- one 50g sensor on the axle in the upward axis
- one 10g sensor on the bumper in the upward axis
- one 10g sensor on the bumper in the forward axis
- one 50g sensor on the bumper in the upward axis
- one 50g sensor on the bumper in the forward axis

Our rationale is that the upward direction will be sufficient for measuring vehicle interactions with the ground. In these cases, the vehicle is mostly vibrating up and down from the ground type. We are looking for these vibrations transmitted through the axle, as well as possibly through the chassis of the vehicle to the bumper unit. The forward direction becomes important when measuring vehicle interactions with above-ground terrain classes. In these cases, as the vehicle drives forward, the bumper interacts in the forward direction with these terrain classes.

The middle left closeup on figure 3.7 is the contact microphone. It is unclear whether this low-end sensor type has an axial sensing direction, but we make the assumption that the main axis is normal to its suction surface. We again mount it in the same scheme:

- one on the axle in the upward axis
- one on the bumper in the upward axis
- one on the bumper in the forward axis

The bottom left closeup on figure 3.7 is the Analog Devices accelerometer, usually found inside IMU's. This is a 3-axis accelerometer. We mount one on the axle and one on the bumper. Each sensor gives three signals for the x,y, and z directions.

## 3.4 LAGR Data Collection

We collected data for many different terrain types, including:

- Ground terrain types
  - grass, grass and leaves, pavement
- Above-ground terrain types
  - soft vegetation, bramble, bushes, trees

We collected data at different locations within the Pennsylvania region that provide the terrain types listed above. These locations included city and state parks; namely, Schenley Park and Raccoon Creek State Park. These locations also included Carnegie Mellon University robotic test facilities; namely the Gascola and Taylor sites.

### Data Collection Procedure

We conducted each trial at a specific location, or what we call a *locale*. We define a *locale* as a unique geographical location, covering some local region where the visual characteristics

of the environment remain similar. For instance, one locale might be the land surrounding a specific parking lot at Raccoon Creek State Park. At each locale, we run at least one data collection trial.

On the Gator we had a human driving the vehicle, but on the LAGR, the vehicle drove autonomously. The LAGR is not a vehicle that can be mounted by a human. It must be operated with a remote if human-controlled. This makes it harder to collect elegant data using human control. Autonomous control is the end goal for a robot anyway, but autonomous control also provides a number of benefits for the quality of the data. One benefit is that a constant speed can be maintained, thereby controlling this variable when considering the proprioceptive data. Another benefit is that we can stop and start the robot at various measured waypoints, allowing us to collect static exteroceptive data at measured distances. This allows us to more easily register the exteroceptive and proprioceptive data. On the Gator we did not use exteroceptive data, so this was not a consideration. Another benefit is that the robot can be steered in a controlled manner towards an area of interest, without training a human to effectively use the remote control (which is surprisingly difficult).

We still used the remote control to drive the robot around in manual mode when it was not in autonomous mode. This allowed us to drive the robot on and off the truck that we used to transport the robot to each locale. This also allowed us to drive the robot out of hazardous situations when it got stuck. And this allowed us to drive the robot to specific waypoints in the beginning of each trial.

For each trial, we had the robot drive in a straight line, stopping every 2 meters so that the Wobbler can make a full revolution. In this way, we get a full 3d point cloud without any motion blur. We chose a straight line as the robot's path to avoid localization errors. When the robot moves, it maintains a constant 0.5 m/s velocity, so that we can control the effect that speed might have on the proprioceptive signals. The sound and vibration signals are pertinent as the robot is moving, and the camera and Wobbler data are pertinent when the robot stops.

For ground classes, the robot can just drive over the terrain without interruption. For above-ground classes, this is not the case. For hard above-ground classes, such as trees and bushes, the robot will stop when it hits the terrain. For semi-hard above-ground classes, such as bramble, the robot will drive through a little bit of the terrain, and then get stuck as the bramble becomes denser. Soft vegetation is the only soft above-ground class where the robot will continue to drive through the terrain for an extended period of time. Since these above-ground classes usually bring the robot to a stop, we have much less data for them than we do for ground classes.

At a specific locale, there is some subset of all of the terrain classes listed above. For instance, a locale might have the classes: bramble, pavement, and grass-leaves. (Note: by grass-leaves, we mean grass and leaves, or rather, dead leaves covering grass.) In this example, there is one above-ground class: bramble, and two ground classes: pavement and grass-leaves. We almost never had more than one above-ground class in a specific trial. Since a trial involves the robot getting stuck in the above-ground class, we set up the robot's path so that this class comes at the end of the path. This way, we did not have to write path-planning and control code that allowed the robot to distract itself from the

bramble. That would have been beyond the scope of this thesis. As mentioned above, the robot's path is a straight line segment. The bramble class is at the end of this line segment.

For each trial, we performed the same procedure to allow the robot to autonomously interact with the terrain classes at the locale. We depict this procedure in figure 3.8. We designed this procedure to be as simple as possible, allowing for a minimal amount of autonomous software design, while still preserving the the benefits of autonomous data collection. For a specific trial, we first remote control the vehicle to the boundary between the one above-ground class and whatever ground class was next to it. We then turn on the robot's path planner to mark this waypoint. This is the robot's final waypoint along the path that it will take. With the robot still in manual mode, we then remote control the vehicle to a spot 10-20 meters away from this final waypoint. This will be the beginning of the robot's straight line path to this final waypoint. The path planner then computes a series of waypoints along this straight line segment, every 2 meters back from the final waypoint. We then switch the robot into autonomous mode, and it proceeds to drive at a constant velocity, stopping at each waypoint to collect a steady point cloud. After stopping at the final waypoint (the boundary to the above-ground class), we then allow the robot to drive for four meters into the above-ground class, but the robot usually comes to a stop before reaching the end of the four meters.

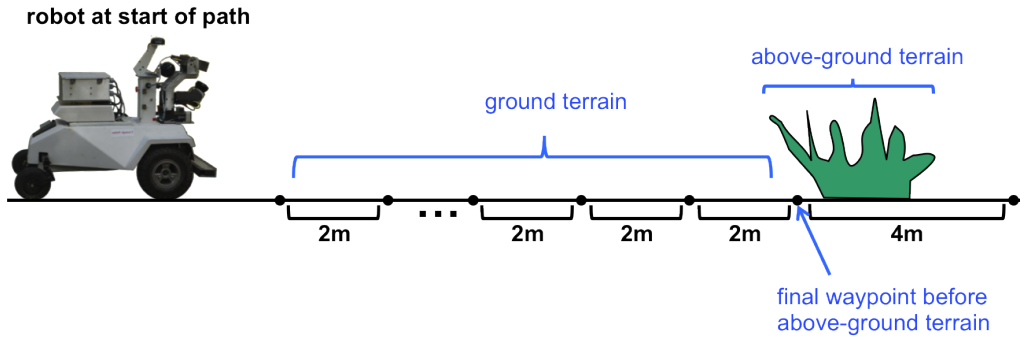


Figure 3.8: Data Collection Procedure

There are several scenarios in which the robot would come to a stop when interacting with the above-ground terrain class. For hard above-ground classes such as trees and bushes, the robot will hit the object with its bumper, and the low-level controller will command the robot to come to a stop by activating the breaks. For semi-hard above-ground classes such as bramble, the robot's bumper will not get hit right away, and the robot will attempt to drive through for a few meters. The bramble will eventually become thicker, which sometimes leads the bumper to be hit, in turn activating the breaks. In other scenarios, the thicker bramble will entangle the body of the robot without the bumper being hit. This will result in the robot continuing to attempt to drive forward without success. This might result in wheel-spinning, but this rarely happens since the robot is commanded at a low speed, and the power of the vehicle is limited. If we had more wheel-spinning data, we might have included this as a class in our classifier.

When the bumpers get activated and the breaks are turned on, the robot automatically

returns to manual mode. When the robot gets stuck without the bumper being hit, we manually switch the robot into manual mode. We then manually drive the robot back out of the above-ground class and prepare it for the next trial.

Lets say again that the three classes that show up at the locale are bramble, grass-leaves and pavement. The robot would interact with the bramble class at the end of the line segment, as the robot drives into the bramble and eventually gets stuck. Along the line segment leading up to the bramble, the robot would interact with the two ground classes, grass-leaves and pavement.

In the beginning of a trial, right after the robot is switched into autonomous mode, before it starts moving along its straight line path, we manually bang on the robot with a rubber mallet near both its bumper and axle proprioceptive sensing suites. This allowed us to manually time synchronize the data in post-processing. This was necessary since each analog-to-digital (A/D) data acquisition board has its own internal clock, and then in turn each embedded computer to which the A/D boards are connected have their own clocks. Some of the embedded computers communicate with time-synchronizing software, but we did not implement this for the Windows Virtual Machine running the National Instruments boards. Even if we had implemented this, the different A/D boards have their own clocks.

### Hand Labeling and Data Point Numbers

Hand-labeling the data involved labeling the proprioceptive data and exteroceptive data separately. The proprioceptive human labels on the training data are used for training the proprioceptive model in the first step of the two-step framework (refer to figure 2.2). Then on the test data, these labels are used for evaluating classifier performance. The exteroceptive human labels are not used for any training in the two-step framework. They are used only on the test data for evaluating performance of the second step in the two-step framework. They are however used for training in other benchmark comparisons, discussed in chapter 4. Chapter 4 describes with much greater care how these labels are used. Table 3.5 lists the locales for which we have labeled data, and which classes occur in each of those locales.

Locale	Classes in that locale
sea2-11	grass, vegSoft
wood1-7	bramble, grass
wood1-8	bramble, grass
bush4-1	bush, grass
tree2-1	treeBig, grass-leaves, pavement
veg2-1	bramble, grass
bramble1-1	bramble, pavement, grass-leaves

Table 3.5: Locales and Terrain Classes in Each Locale

The proprioceptive data is time-series data from the sound and vibration sensors. Since these signals are all synchronized, we only have to label one signal. Our approach for

labeling this data is similar to how we labeled the data on the Gator platform. Just as we used the webcam on the Gator as a sanity check to guide us in the labeling, now we use the image stream from the Pixim cameras. We use interactive software that we developed to go back and forth between time series plots, the audio playback of the signals, and the camera image streams. For above-ground classes, the time-series plots are particularly helpful because there is a large increase in the signal energy, evident in the plots. This is especially true for the more sensitive sensors, as well as the sensors mounted to the bumper. Our software allows us to alternate between the time series plots of all of the time-series signals, which at this point have all been synchronized in post-processing from the rubber mallet bangs. So depending on what class we are trying to label, we might choose to examine one particular signal or another. For the ground classes, labeling was much easier, as these events covered very large spans of time. The robot stops every two meters at a waypoint. In between the waypoints, while the robot is moving, we collect the vehicle-terrain interactions. We call this a *moving interval*. We only use the part of the moving interval during which the robot has achieved a constant velocity of 0.5 m/s. Since velocity is an output we get from the LAGR’s low-level control, we can automatically find the start and end times of these intervals. So labeling this data only involves the human labeling the entire moving interval as a specific terrain. In this way, one can see our approach for the first step of the two-step framework as semi-supervised instead of supervised. We are still, however, doing fully supervised labeling for the above-ground terrain classes. We simply ignore moving intervals that are on the boundary between two ground classes. We could always come back to this data later, but we have plenty of ground data, so it wasn’t necessary to label every last window for training. Note that when the proprioception is making predictions for teaching the exteroception, this boundary data is still used. Section 5.2 discusses hand-labeling of the proprioceptive data in more detail.

After we completed the labeling, we had sequences of time series data where each sequence represented an event of interacting with a specific class. As we did on the Gator, we split these sequences up into 200ms short time windows, with 50% overlap between successive windows. (Chapter 5 discusses the pipeline for turning a data point window into a class prediction.) Table 3.6 gives a list of how many human-labeled data points (short time windows) we have for each terrain class in each locale.

Locale	Class	No. data pts
sea2-11		
	grass	296
	vegSoft	132
wood1-7		
	bramble	29
	grass	175
wood1-8		
	bramble	65
	grass	198
bush4-1		
	grass	148
	bush	11
tree2-1		
	grass-leaves	102
	pavement	24
	treeBig	2
veg2-1		
	grass	127
	bramble	42
bramble1-1		
	bramble	41
	grass-leaves	49
	pavement	25

Table 3.6: Number of Data Points for Proprioceptive Data

The exteroceptive data is the image stream coming from the two Pixim cameras, along with the point cloud coming from the Wobbler. At each waypoint, the robot stops for three seconds to allow the Wobbler to make a full revolution. We call this a *stopped interval*. We only use the exteroceptive data from the stopped intervals. Although there is a continuous stream of images collected during the trial, we only use one image pair from each stopped interval. By image pair, we are referring to the images from the left and right cameras. And we are only using the subset of Wobbler points within these stopped intervals. Considering the start and end timestamps of a stopped interval, we calculate the midpoint timestamp, and then consider the pair of Pixim images closest to this midpoint. We then consider a three second window centered at this midpoint, and take the subset of Wobbler points within that window. So considering again figure 3.8, for each waypoint along the path, we use one pair of images, and the associated three second point cloud around those images. For the subset of Pixim images that we are considering, we then hand-label the terrain classes in the images. We use the Gimp image editor, and we paint polygonal regions for each terrain. We use a unique color for each terrain class that can be identified by our software for processing. The associated Wobbler point cloud for that image pair is

projected onto the image plane. So then there are a subset of pixels within the image pair onto which Wobbler points are projected. For each of the pixels in this subset, we consider a locale image patch centered around the pixel. These are the exteroceptive data points that we consider for self-supervised learning. These image patches take on the human label of the center pixel, as well as the 3d (x,y,z) location of the Wobbler point associated with that center pixel. These (x,y,z) locations allow these data points to be registered within a larger world map. This larger map allows for registration with the proprioceptive data, as well as other future mapping applications. Chapter 4 discusses data registration in more detail. Chapter 6 discusses the pipeline for turning an image patch data point into a class prediction. For now, we just list the number of human-labeled data points we have for the exteroceptive data in table 3.7.

<b>Locale</b>	<b>Class</b>	<b>No. data pts</b>
sea2-11		
	grass	21405
	vegSoft	15300
wood1-7		
	bramble	9032
	grass	19750
wood1-8		
	bramble	12888
	grass	19113
bush4-1		
	grass	21102
	bush	13106
tree2-1		
	grass-leaves	20625
	pavement	20890
	treeBig	2481
veg2-1		
	grass	20537
	bramble	9735
bramble1-1		
	bramble	4541
	grass-leaves	20940
	pavement	12744
	treeBig	690

Table 3.7: Number of Data Points for Exteroceptive Data





# Chapter 4

## Self-Supervised Framework

In this chapter, we discuss the framework for how we use proprioception to train models for exteroception in self-supervised schemes. We consider the frameworks presented in figures 2.1 and 2.2 to be the two most general frameworks for self-supervised learning. From this generalization, we look at the specific cases where proprioception is the teacher and exteroception is the student. The proprioception is interacting with the terrain to estimate properties about the external environment. These estimates are then used to train the exteroception on that same environment. In the first framework (figure 2.1), the proprioception can directly provide estimates. The advantage with this framework is that no human estimates are needed for the property values. The disadvantage is that the properties that can be estimated are constrained to direct first principles pertaining to the sensed interaction. In the second framework (figure 2.2), there is an initial supervised preprocessing step where the proprioception is trained on a model, and then in the second main self-supervised step, the predictions from this model are used to train the exteroception. We call this initial supervised step a preprocessing step because the assumption is that only a small amount of human-labeled data needs to be used to train this model. As discussed earlier, our hypothesis is that proprioceptive features vary less from one location to another, compared with exteroceptive features, so their models can be trained on a small number of locations and still generalize well.

Our work builds off the second framework in figure 2.2, so that there are fewer limitations on the properties estimated. We show this same diagram below in figure 4.1 for convenience. In this framework, proprioceptive features come from acoustics and vibration data, as discussed in chapter 5. Exteroceptive features come from camera imagery, as discussed in chapter 6. Ladar range data is used to register the proprioceptive and exteroceptive data. The *properties* boxes in figure 4.1 include a set of semantic categories that describe the terrain. These include:

- Ground terrain types
  - grass, grass and leaves, pavement
- Above-ground terrain types
  - soft vegetation, bramble, bushes, trees

To distinguish between these multiple categories, we implement multiclass classifica-

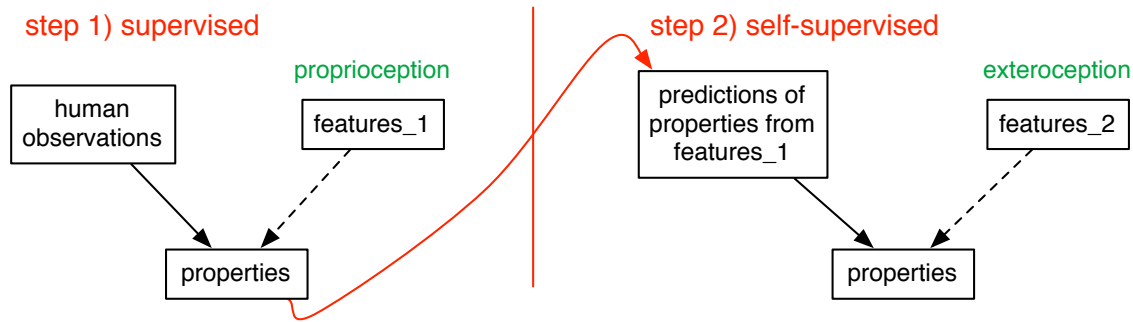


Figure 4.1: General two-step framework. In the first step, supervised learning is used to train one set of features. In the second step, the model trained from the first set is used to train a second set of features. (This is a repeat of figure 2.2, placed here for convenience.)

tion. A supervised multiclass classification algorithm is used for each of the two steps in figure 4.1. Note that even though the second step is self-supervised, we still use a supervised algorithm. These algorithms expect training data labeled by a teacher but it does not matter whether this teacher is human. Separate models are trained for each of the two steps in the framework. Chapters 5 and 6 discuss the supervised models for the proprioception and exteroception modules, respectively. This chapter is concerned with the technicalities of combining these models into a full self-supervised scheme. We introduced the data collection on the LAGR platform in section 3.4. We show in this chapter how this data is processed to implement the self-supervised framework within our system. Section 4.1 explains the registration of the proprioceptive and exteroceptive data. Then, once the data has been collected and registered, we explain in section 4.2 how we implement our self-supervised framework. We then present the main experiments that we run on our data in section 4.3. In section 4.4, we conclude the chapter by revisiting the thesis problem, and show how the main experiments will be used to prove our hypotheses.

## 4.1 Data Registration

In this section we discuss how we spatially register the proprioceptive and exteroceptive data. Figure 4.2 depicts the coordinate frames in block diagram form which are used for data registration. We will refer to these in turn as we describe the registration.

When the robot is turned on at a new locale, it is at the origin of its local coordinate frame. A static local coordinate frame is defined in this way for each new locale. The robot continuously outputs global pose estimates from its GPS. The one global pose estimate at the local origin timestamp is all we need to transform any data in the local map into the world frame, using a homogeneous transformation matrix. For the rest of this discussion, we just consider the local frame.

In addition to global pose estimates, the robot also continuously outputs 3d local pose estimates ( $x, y, z, roll, pitch, yaw$ ). These estimates are produced by the low level control

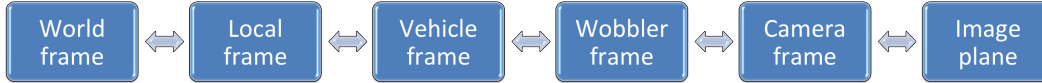


Figure 4.2: Coordinate Frame Block Diagram. All of the coordinate frames that must be considered when for data registration.

from the original LAGR system, generated from its Extended Kalman Filter (EKF). We tested the accuracy of this filter. When running on flat surfaces indoors or on level flat outdoor terrain, this filter works very well, with under 2% of drift error over considerable distances. But in most of the outdoor off-road environments where our experiments are conducted, this filter fails significantly. Furthermore, there is very little documentation on how this pose is generated, making it hard to reverse engineer. Instead, we generated our own 2d point and shoot estimate  $(x, y, yaw)$ . The heading from the gyroscope and the distance from the wheel encoders are used to generate this estimate. We make a 2d planar assumption for any locale in which we operate. This still works if we are on the side of a hill, as long as the stretch of the hill we are running on is planar. (Such hill sides are very common in our data collection sites.) Because we are only conducting our trials over 10-20 meter long paths, this local assumption is manageable, without having to worry about terrain undulations. Point and shoot (dead reckoning) estimates are notorious for building up significant drift from heading errors, but since our data is collected along a straight line path for short distances, we do not have to worry about this. We also avoid most wheel slip since our robot drives at low speeds.

The origin of the vehicle frame is at the center of its front wheel axle, projected onto the ground. The 2d vehicle pose estimate  $(x, y, yaw)$  is the pose of the vehicle frame’s origin with respect to the local frame. The local frame and the vehicle frame are both in North-East-Down (NED) coordinates. The forward (north) axis is the x-axis. The sideways (east) axis is the y-axis. The downwards axis is the z-axis. The  $x - y$  plane is the floor of the local map. The vehicle frame origin is always along this floor. The terrain being classified can come up off the floor, and the point cloud from the Wobbler gives the  $x - y - z$  point cloud of this terrain. The 2d  $(x, y, yaw)$  pose of the robot at any point along its path, along with zero values for  $(z, roll, pitch)$ , are used to transform data between the vehicle frame and the local frame through a homogeneous transformation matrix.

As discussed in section 3.4, the data collection trials involve driving the LAGR in a straight line, stopping every 2 meters so that the Wobbler can make a full revolution. Figure 3.8 in that section depicts this procedure. Each waypoint along the path denotes a *stopped interval*, where the robot collects exteroceptive data. Each segment between two waypoints denotes a *moving interval*, where the robot collects proprioceptive data of either a ground terrain interaction or above-ground terrain interaction.

In figure 4.3 we depict one of these stopped intervals. In figure 4.4 we depict one of these moving intervals. The moving interval considered is always at a later time than the stopped interval, so that it is in front of the robot when the robot is at the stopped interval. The forward x-axis is the straight line being shown in figures 4.3 and 4.4.

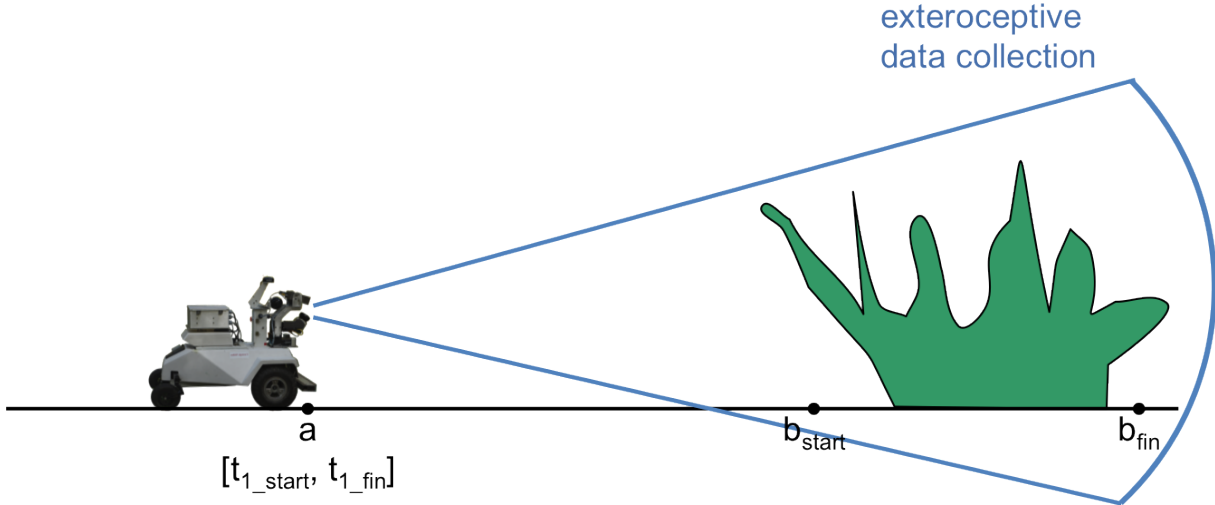


Figure 4.3: Robot at Some Stopped Interval, Collecting Exteroceptive Data. Stopped at pose  $a$ , the robot collects exteroceptive data over the time interval  $[t_{1Start} , t_{1Fin}]$ . It collects this data for the spatial interval  $[b_{start}, b_{fin}]$  in front of it.

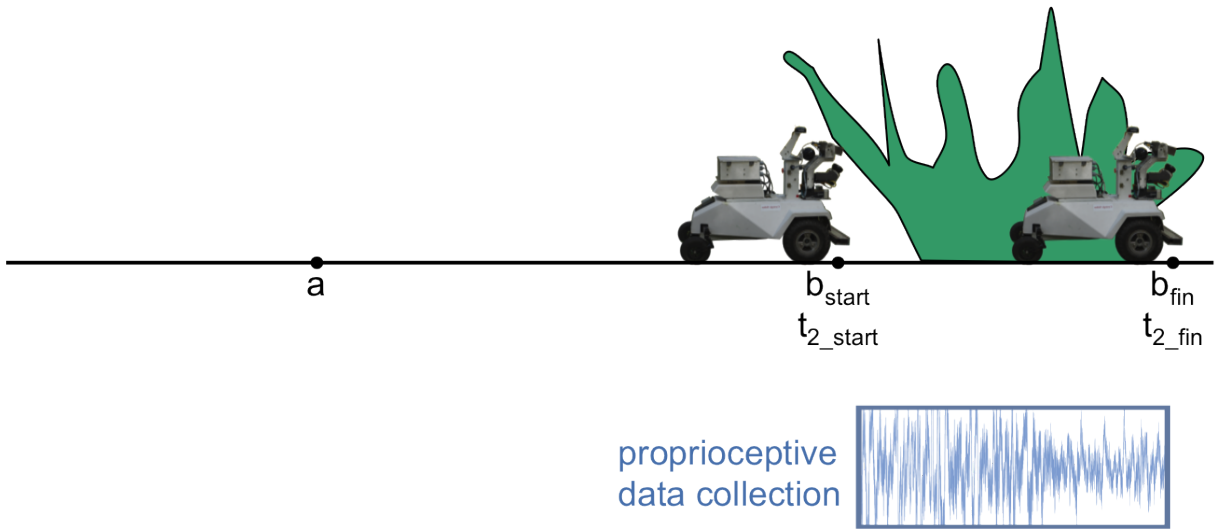


Figure 4.4: Robot During a Moving Interval, Collecting Proprioceptive Data. As the robot moves over the spatial interval  $[b_{start}, b_{fin}]$ , during the time interval  $[t_{2\_start} , t_{2\_fin}]$ , it collects proprioceptive data.

First lets consider figure 4.3. During the stopped time interval  $[t_{1\_start} , t_{1\_fin}]$ , the

robot is at pose  $a$  in the local frame. From this pose, it sees all of the terrain in front of it. This time interval  $[t_{1Start}, t_{1Fin}]$  is greater than or equal to three seconds, allowing the Wobbler to make a full revolution, thereby collecting a point cloud without any motion blur. The origin of the Wobbler frame is between its roll and tilt motor axes. The SICK LMS 151 gives a planar fan of time-of-flight readings using its laser and internal mirrors, giving range readings at every 0.5 degrees, over a 270 degree field of view, centered around the forward axis. These polar range and angle coordinates can be converted to Cartesian x-y coordinates within the scan plane of the SICK, and then together with the roll and tilt angles of the Wobbler’s motor, and the distance from the SICK scan plane to the Wobbler origin, these values can be turned into x-y-z points in the Wobbler frame. The pose of the Wobbler frame with respect to the vehicle frame is fixed, and the associated homogeneous transformation matrix can be used to transform the  $x - y - z$  point cloud data from the Wobbler frame to the vehicle frame. And then this data can in turn be transformed to the local frame. (Refer to figure 4.2 to keep track of coordinate frames.) Moving forward with this discussion, we consider the point cloud in the local frame.

In figure 4.4, we consider the time interval  $[t_{2.start}, t_{2.fin}]$ , where the robot is moving forward along the x-axis from  $[b_{start}, b_{fin}]$ . Poses  $b_{start}$  and  $b_{fin}$  are again in the local frame, and they denote the start and end of the particular terrain of interest.

Of course the terrain of interest is not just a line segment from  $b_{start}$  to  $b_{fin}$ . It is an entire volume. We can consider a rectangle along the  $x - y$  floor of the local map over which the vehicle interacts. The length of the rectangle is from  $b_{start}$  to  $b_{fin}$ , and the width of the rectangle is the width of the robot between its left and right front wheels. The width is along the y-axis, which in figures 4.3 and 4.4 would be coming in and out of the page. The x-axis is through the center of the length of the rectangle. We then consider a rectangular prism extruded infinitely upwards from this rectangular region along the ground plane. Considering the point cloud from  $[t_{1Start}, t_{1Fin}]$ , which has been transformed to the local frame, we can look at the subset of points that fall within this rectangular extrusion. We now have a concept of the volume of terrain with which the robot is interacting.

The proprioceptive data makes it easy to label this terrain. We show a proprioceptive signal in blue beneath this terrain interaction in figure 4.4. By providing human labels in the time domain for the start and finish of the vehicle-terrain interaction, namely  $[t_{2.start}, t_{2.fin}]$ , we then have a terrain label for the line segment from  $b_{start}$  to  $b_{fin}$ , since our pose estimates have associated timestamps. And then in turn we have a terrain label for the volume of terrain described above. Note that the human only has to mark two points along one dimension, the time axis, to label this terrain. Without the help of proprioception and this data registration process, the human would have to label an entire area or volume in the 3d world or the image plane. Also note that we considered here the above-ground terrain interaction sequence. The ground terrain that comes before it along the robot’s path can also be conceptualized in a similar manner.

In figure 4.5 we show the same proprioceptive time series sequence that we showed in figure 4.4, this time split up into short time windows. These short time windows are the proprioceptive data points. If we have human labels for the start and finish of the entire sequence, as we do for our training data, then the short time windows take on the label of the sequence. If we are considering test data, then each window will have a predicted label,

and there will be no overarching label for the entire sequence. So instead of considering the volume of terrain over the entire sequence from  $[t_{2\_start} , t_{2\_fin}]$ , we consider separate volumes of terrain for each short time window.

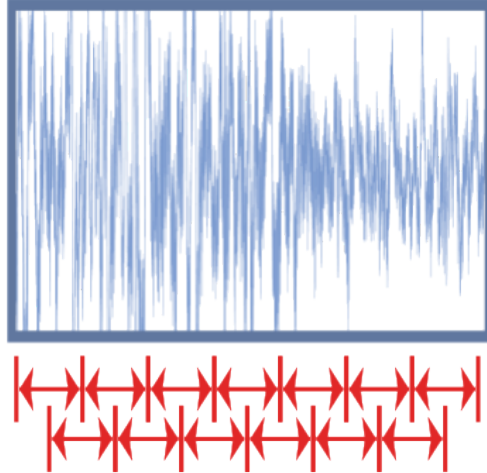


Figure 4.5: Time Series Signal Representing an Instance of the Robot’s Interaction with a Particular Terrain Class. This is analogous to the signal shown in figure 4.4, representing a sequence from  $[t_{2\_start} , t_{2\_fin}]$ , where the vehicle interacts with a specific terrain class. The sequence is split up into consecutive overlapping short time windows, which are then proprioceptive data points.

The process for computing the terrain volume is the same. Instead of the start and finish timestamps being for the entire sequence, they are for the start and finish of a short time window. These correspond to start and finish poses along the forward x-axis. Again using the width of the robot’s front wheel axle, we compute a rectangle along the  $x - y$  floor of the local frame. We refer to this short time window rectangle as a robot *footprint*. We then again extrude a vertical prism upwards, and consider the subset of points from the Wobbler point cloud in the local frame that fall within this prism. This subset of points takes on the terrain label of that short time window. The label might be from a human, or it might be from the proprioceptive model prediction.

Referring again to figure 4.3, with the robot at stopped interval  $a$ , we can consider the 3 second point cloud at that interval, along with any pair of images from the left and right cameras. An example image pair from a stopped interval is shown in figure 4.6. The two cameras together cover about a 72 degree field of view in front of the robot, with each camera covering about half the field of view to either side of the forward axis. The two cameras are slightly cross-eyed. So the left camera image is on the right of this diagram, and the right camera image is on the left. We show a little bit of black negative space between the two images here just to keep the images separate, but there is no gap in the field of view of the robot. There are a few pixel columns of overlap in the middle that show the same terrain in both images.

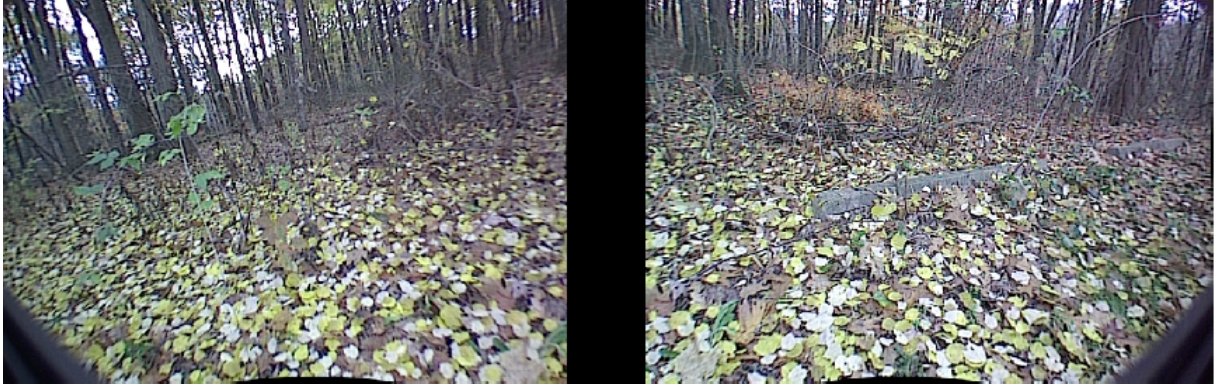


Figure 4.6: A Pair of Images from the Left and Right Pixim Cameras. Note that the black column down the middle just separates the images for clarity. There is no gap in the forward field of view. There is a slight overlap in the field of view.

We have considered so far the Wobbler point cloud transformed from the Wobbler frame to the vehicle frame to the local frame, in order to associate it with a short time window rectangular prism. Referring to the coordinate frame block diagram in figure 4.2, this is taking the point cloud data from the Wobbler frame and moving it to the left along the pipeline over the local frame. We now consider moving the point cloud data to the right, from the Wobbler frame to the camera frame to the image plane. The pose of the camera origin with respect to the Wobbler origin is fixed, and a homogeneous coordinate transform is used to go from one frame to the other. Then from the 3d camera frame, we go to the 2d image plane of each camera using their associated intrinsic camera matrices. All of these transform matrices were computed during the camera-Wobbler calibration process. The Wobbler points are hence projected onto the image plane of each camera. Remember that the image pair in figure 4.6 was from the stopped interval  $a$  in figure 4.3. The three second Wobbler point cloud from that same stopped interval is shown in figure 4.7, projected onto each image in that same image pair. Each red point is a projected Wobbler point.

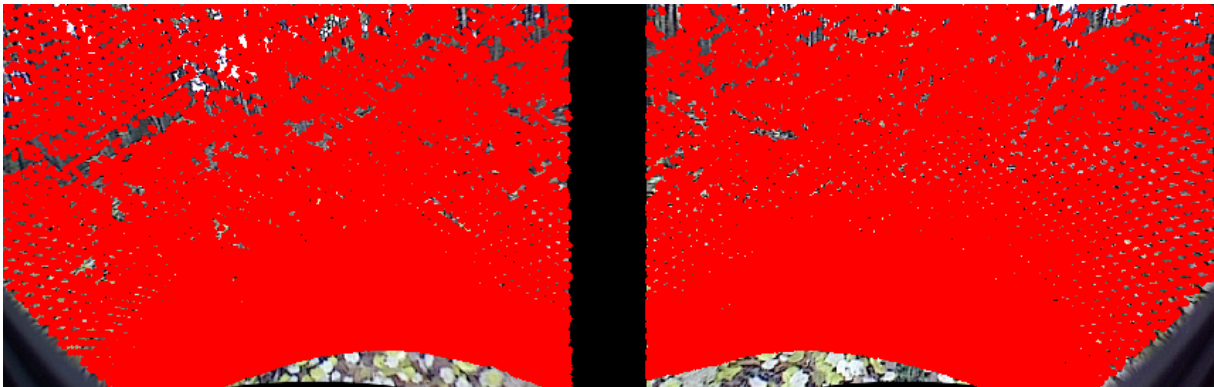


Figure 4.7: A Pixim Pair from a Stopped Interval, with the 3-second Wobbler Point Cloud Projected onto Them. An exteroceptive data point is then a local image patch centered around each projected Wobbler point.

Similar to how the Wobbler points are projected onto the image plane, we also consider the robot footprints projected onto the image plane. Each robot footprint is a rectangle on the  $x - y$  floor of the local frame, with  $z = 0$ . Each corner of the rectangle is then an  $(x, y, z)$  point in the local frame, and is propagated all the way through the pipeline of figure 4.2 to the image plane. Looking back at figure 4.3, we consider the entire path in front of the robot, from pose  $a$  to pose  $b_{fin}$ . This encompasses both the ground and above-ground terrain. In the time domain, this is the interval from  $t_{1\_fin}$  to  $t_{2\_fin}$ . We can split this entire interval up into short time windows, and in turn consider them as rectangular footprints along the local frame floor. (The only footprints that will have a non-zero area are the ones where the robot is moving.) These footprints are then projected onto the image plane of each camera in figure 4.8. Note that for data registration purposes, the footprints are just used in the original local frame. But for conceptual purposes, we project them here onto the image plane. The union of these rectangles show the robot's path in the image plane.

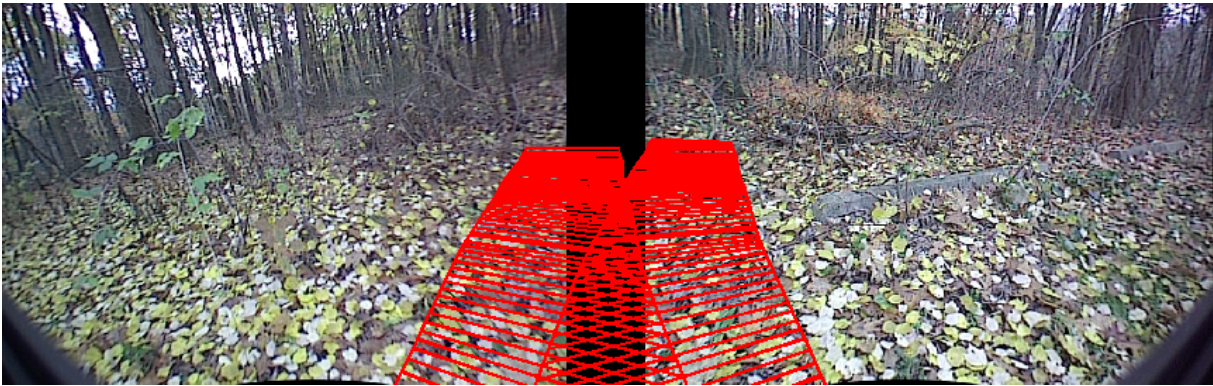


Figure 4.8: Rectangular Footprints Projected onto the Images, Showing the Robot's Path

Now we can consider the subset of Wobbler points that fall inside the robot's path. Remember that for each rectangular footprint, we extrude a vertical prism upwards and consider the subset of Wobbler points that fall within that prism. So each footprint has a point cloud subset associated with it. Figure 4.9 shows the union of the point cloud subsets for each of the rectangles in figure 4.8. In other words, this is the set of points that are inside the robot's path, projected onto the image plane. Each subset of points takes on the terrain label of its associated footprint. Remember that exteroceptive data points are local image patches centered around each projected Wobbler point. Each local image patch then has a label coming from its Wobbler point, which in turn got its label from its associated footprint. When the footprint labels are coming from the proprioceptive predictions, we then have self-supervised labels for our exteroceptive data. The red points in this figure (the points inside the robot's path) therefore represent the training data for the exteroceptive module of our self-supervised framework. Referring back to figure 4.1, this is training data for the second step.



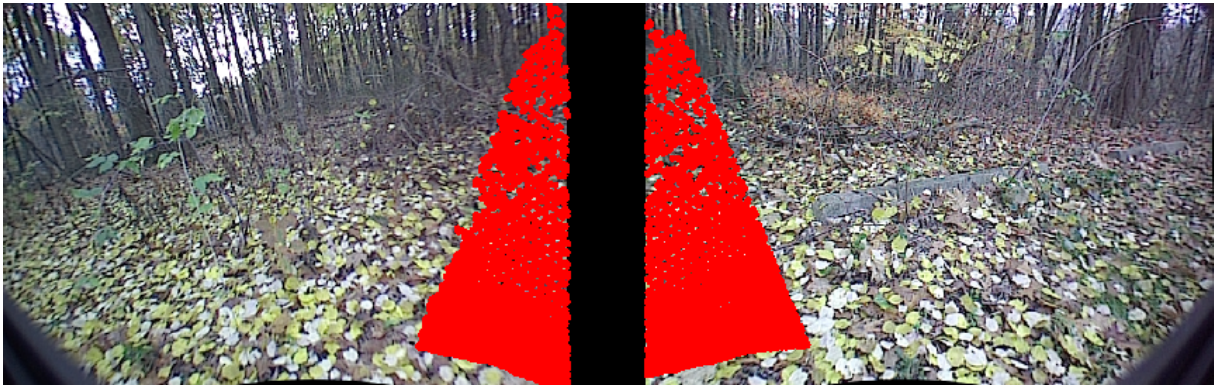


Figure 4.9: Wobbler points that are Inside the Robot’s Path. These are the subset of Wobbler points from figure 4.7 that fall within the rectangular footprints of the robot’s path shown in figure 4.8.

Referring back to figure 4.7, which is the full set of points from the point cloud at that stopped interval, we consider the points inside the robot’s path as a subset of this full set. We then take the set complement to get the points outside the robot’s path. These points are depicted in figure 4.10. These points are not associated with robot footprints, so they do not have proprioceptive labels. However, they cover a wider region of the map. They are everything the robot sees within the locale that it has not interacted with. This is the test data for the second step in figure 4.1. Or rather, the test data is the set of local image patches surrounding each of the red points.

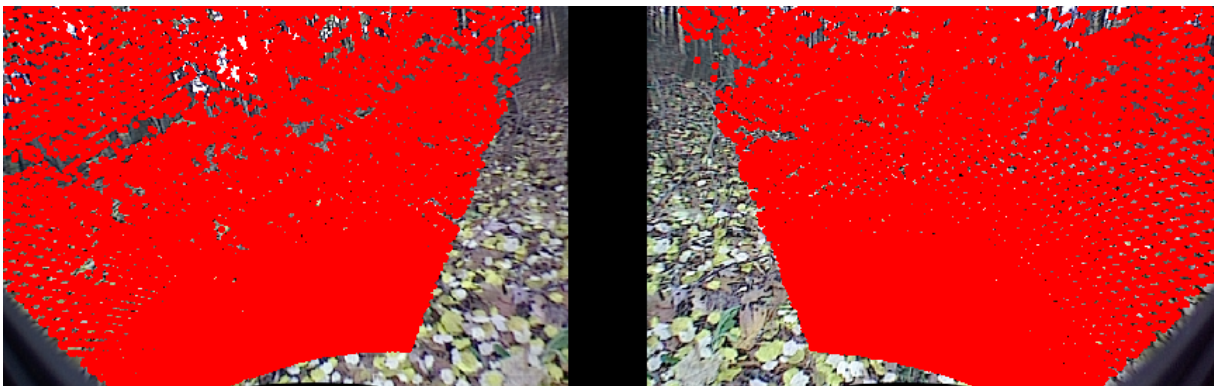


Figure 4.10: Wobbler points that are Outside the Robot’s Path. These are the subset of Wobbler points from figure 4.7 that do *not* fall within the rectangular footprints of the robot’s path shown in figure 4.8.

## 4.2 Applying the General Framework To our System

Figure 4.11 shows again the points inside the robot’s path on the top left and the points outside the robot’s path on the top right. This is the training data and the test data for

the self-supervised step, respectively. On the bottom is a diagram that conceptualizes this setup within the overall theme of the thesis. The perspective in this diagram is from a bird's eye view. The page is the  $x - y$  ground plane of the local map, and the  $z$ -axis is into the page.

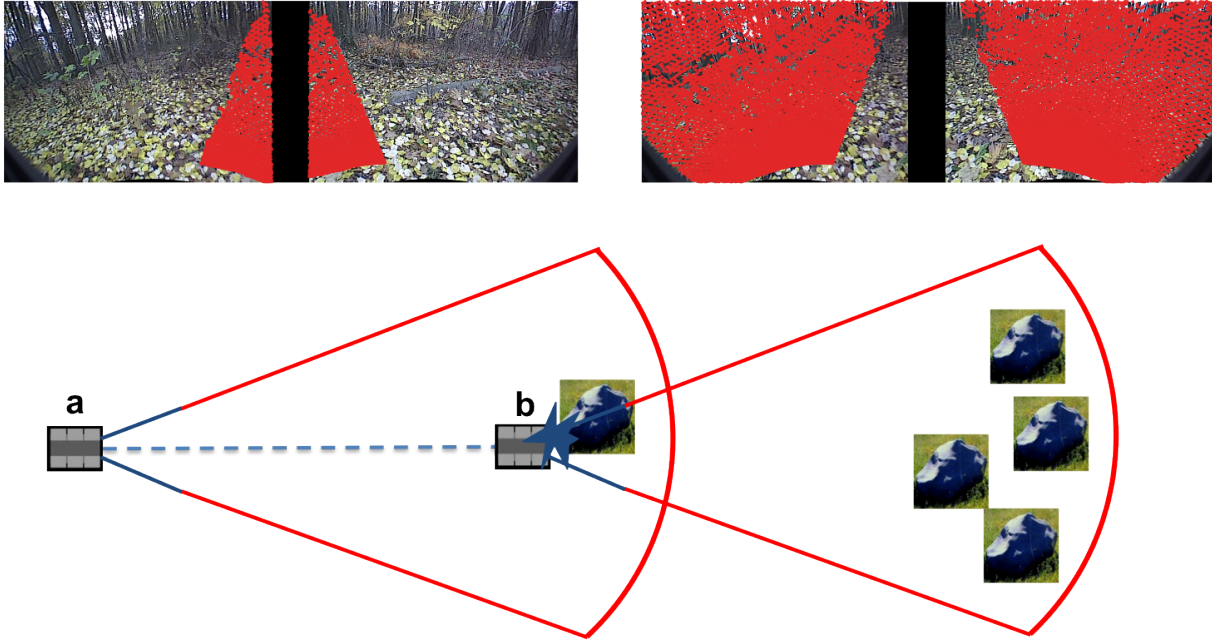


Figure 4.11: Relationship between Theoretical Diagram and Actual Data. *Top Left*: Actual exteroceptive data inside the robot's path. *Top Right*: Actual exteroceptive data outside the robot's path. *Bottom*: Theoretical Diagram for the robot collecting proprioceptive data, exteroceptive data inside the robot's path, and exteroceptive data outside the robot's path.

The left side of the diagram on the bottom of figure 4.11 is similar to figures 4.3 and 4.4. The robot on the far left is the robot at the stopped interval at pose  $a$  in figure 4.3. In this example, the terrain being considered are rocks. The left red cone is like the cone in figure 4.3, showing what the robot sees from this pose. The robot figure in the middle that is hitting the rock is the robot later in time, analogous to figure 4.4. The crash icon between the robot and the rock is what the robot feels, and this is the proprioceptive time series data.

The rock inside the left red cone in the diagram on the bottom of figure 4.11 represents the exteroceptive data for which the robot will subsequently have proprioceptive labels. This is data that the robot both *sees* and *feels*. It is the data inside the robot's path, as shown by the red Wobblers points on the top left pair of images in figure 4.11. This can then be self-supervised training data.

The multiple rocks in the red cone on the right of the diagram on the bottom of figure 4.11 represent the exteroceptive data outside the robot's path. This is analogous to the red Wobblers points on the top right pair of images in figure 4.11. This data covers a

larger region of the map, and the exteroceptive model can give better predictions for this terrain once it has been trained on the data inside the robot’s path.

So in summary, the robot *feels* or interacts with a small subset of terrain within the locale (left image pair and left cone in figure 4.11). This data is used for training the self-supervised model. And then the robot can make predictions on a larger set of terrain within the locale using its newly trained exteroceptive model (right image pair and right cone in figure 4.11).

Note that in figure 4.11 we are depicting the training set and test set as one pair of images, but in reality there are many pairs of images for a particular locale. From each waypoint along the robot’s path in figure 3.8, we are at a stopped interval, where an image pair and an associated point cloud on that image pair can be acquired. Each of those image pairs are split up into data inside the robot’s path (training data) and data outside the robot’s path (test data). So if we have 10 waypoints for a particular path at a locale, then we will have 10 image pairs, and 10 sets of data inside and outside the robot’s path. We take the union of the sets inside the robot’s path for our training data, and the union of the sets outside the robot’s path for our test data.

Note that this rock example is a hard above-ground impact instead of a soft one, so the robot stops when it makes impact. So instead of  $b_{start}$  and  $b_{fin}$  like we had in figure 4.4, we just have  $b$  in figure 4.11. Or rather  $b_{start} = b_{fin}$ , because the robot is not moving. The impact with the rock would happen over a very short period of time. We might have just one short time window that spans the impact in the proprioceptive time series data. When this window is converted to a rectangular footprint on the  $x - y$  floor of the local map, the rectangle would have no area because  $b_{start} = b_{fin}$ . In turn, the rectangular prism extruded upwards would have no volume, and so the points from the point cloud that hit the surface of the rock would not fall into this prism.

To handle this case, we extend the length of such footprints by 0.3 meters. If the robot hits a hard above-ground terrain type, it will predict from its proprioceptive model that this has happened, and then the rectangle associated with this data point window will be extended in length by 0.3 meters along the forward axis of the vehicle. This allows the hard obstacle to be encompassed by the extended rectangle. The robot will never actually drive through the hard obstacle, so its actual footprint will never overlap with it. The obstacle will always be in front of the robot, not underneath it. By extending the rectangle of the footprint in this way, we allow the data to still be associated correctly.

Now we introduce some graphical notation, so that we can refer easily to what we just described. The graphic on the right in figure 4.12 will be used to denote data *inside the robot’s path*. The graphic on the right in figure 4.13 will be used to denote data *outside the robot’s path*. For either graphic, the encompassing rectangular box denotes a set of locales, with each locale containing a set of image pairs from each waypoint. The red region in figure 4.12 is the set of data inside the robot’s path for that set of locales. The red region in figure 4.13 is the set of data outside the robot’s path for that set of locales. We will also use the *inside the robot’s path* graphic for discussing the purely proprioceptive data, even though technically this graphic is representing the image plane. So, in other words, if we are considering the proprioceptive signals from a set of locales, we will use the graphic in figure 4.12. For proprioceptive data, the data is always inside the robot’s path, because

the robot must interact with the terrain in order to acquire data.



Figure 4.12: Inside the Robot's Path. *Left*: Wobbler points inside the robot's path, projected onto the image plane. *Right*: Corresponding short-hand graphic to denote *inside the robot's path*.



Figure 4.13: Outside the Robot's Path. *Left*: Wobbler points outside the robot's path, projected onto the image plane. *Right*: Corresponding short-hand graphic to denote *outside the robot's path*.

In figure 4.14 we relate this graphics notation to our general framework. We always use blue to denote training data and orange to denote test data. For the blue training data, when human labels are used, we mark this with a blue human icon. When proprioceptively-predicted labels are used, we mark this with a blue microphone icon. On the left, we have the supervised proprioceptive learning step. Here we use data from previous locales for training, and data from the current locale for testing. Note that the training and test data here are purely in the proprioceptive time domain. The data is always inside the robot's path, because the robot must interact with the terrain in order to acquire proprioceptive data. On the right, we have the self-supervised step. Both the training and test data are at the current locale. The training data is inside the robot's path and test data is outside the robot's path. The labels from the training data come from the proprioceptive predictions in the previous step, as shown with the black arrow.

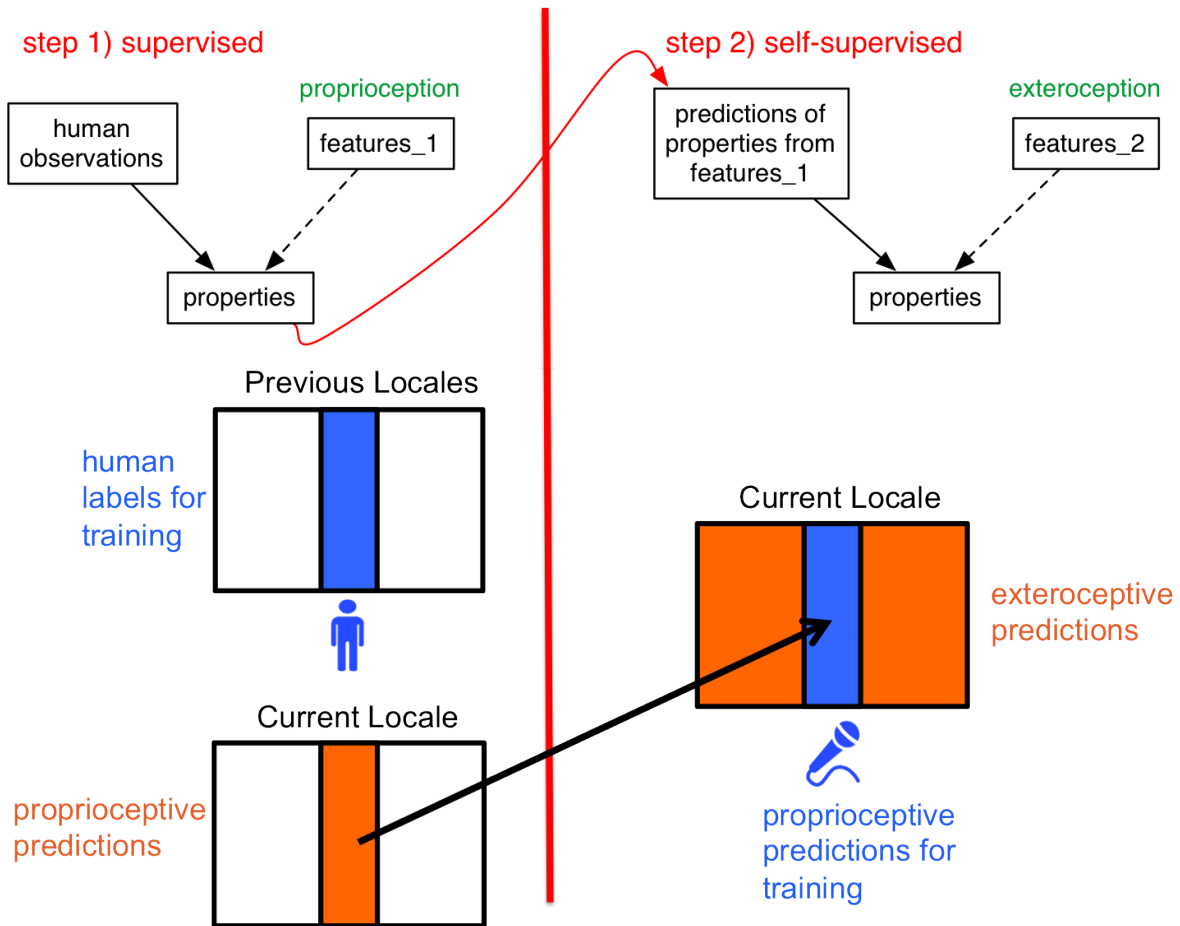


Figure 4.14: Relationship between Theoretical Framework and Graphics Notation. *Top*: Self-Supervised Framework, as presented in figure 4.1. *Bottom*: Graphics notation to demonstrate how this framework is implemented in our system.

### 4.3 Four Main Experiments

In tables 4.1 and 4.2 we encapsulate the two steps of this self-supervised framework. This is a summary of everything discussed so far. Moving forward, this lays the groundwork for the experiments and results discussed in chapter 7.

Table 4.1 is the first step of the two-step self-supervised framework (the left-hand side of figure 4.14). The data stream is time series signals coming from the microphones and vibration sensors. The stream is split up into short time window data points. The training data is human labeled. The start and finish of each terrain interaction signal is marked, and that sequence is given a semantic terrain class, one of  $\{A, B, C, D\}$ . In reality, it is given one of seven class labels but we just use  $\{A, B, C, D\}$  for generality. The training data comprises one or more locales. Enough locales are used to cover the entire set of classes. (Each locale will have some subset of the classes.) The set of locales used is referred to as the set of *previous locales*. The multiclass model is then trained in a supervised fashion.

The test data then comes from a different locale, and we refer to this as the *current locale*. There can be more than one current locale, but each one is considered separately. At each current locale, the second self-supervised step is trained and tested. Human labels of the proprioceptive data at the current locale are acquired, but these labels are only used for evaluating the accuracy of the proprioceptive model. They are not used for training the exteroceptive data.

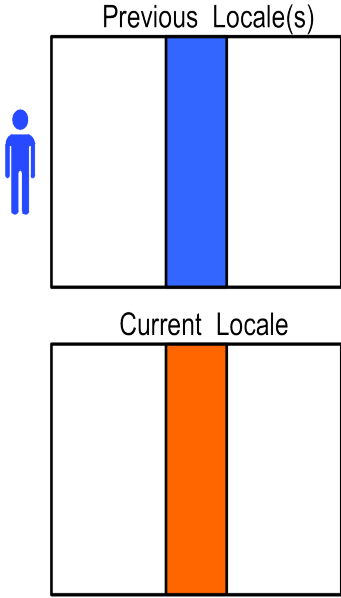
<ul style="list-style-type: none"> <li>• data source: time series signals from microphones and vibration sensors</li> <li>• data point format: consecutive overlapping short time windows from the time series signals</li> <li>• multiclass model: classes <math>\{A, B, C, D\}</math> (in reality, 7 classes)</li> <li>• training data             <ul style="list-style-type: none"> <li>▪ labels for training: human (marked sequences in time series signal)</li> <li>▪ location: previous locale(s), inside path</li> </ul> </li> <li>• test data             <ul style="list-style-type: none"> <li>▪ labels for evaluation: human (marked sequences in time series signal)</li> <li>▪ location: current locale, inside path</li> </ul> </li> </ul>	
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------

Table 4.1: Proprioception Overview

Table 4.2 is the second step of the two-step self-supervised framework (the right-hand side of figure 4.14). The data source is now the stream of camera images coming from the left-right pair of cameras. One image pair is extracted from each waypoint where the robot stops to collect a Wobbler point cloud. The subset of pixels onto which Wobbler points are projected is then used. The data points are then local image patches centered around each projected Wobbler point. The training data is the subset of these projected Wobbler points that fall inside the robot’s path. The labels for the training data come from the proprioceptive model’s predictions. Each proprioceptive data point (short time window) has a prediction, which gets mapped into the local frame as a rectangular footprint. Each Wobbler point that lies above a footprint gets tagged with the predicted label for that footprint. Then when the Wobbler points are mapped into the image plane, their associated labels come along with them, and so each local image patch inside the robot’s path has a label. The current locale will only have a subset of the classes in the multiclass model. We denote this as the subset  $\{A, B, C\}$  out of the full set  $\{A, B, C, D\}$ . This subset of classes is determined by the proprioceptively predicted labels inside of the robot’s path. The proprioception could have false positives, leading to an erroneously larger subset, or it could have false negatives, leading to an erroneously smaller subset. These errors will be

propagated through the exteroceptive model. Even if the proprioception is perfect, there could be some classes that exist outside of the robot’s path that do not exist inside of the robot’s path, and these errors will also be propagated through the exteroceptive model. The test data is the subset of projected Wobbler points that fall outside the robot’s path, and then in turn the local image patches centered around these projected points. These data points do not have proprioceptively-predicted labels. Human labels for the test data are acquired by painting in Gimp directly on the image plane. The human labels are only used for evaluating the accuracy of the exteroceptive model.

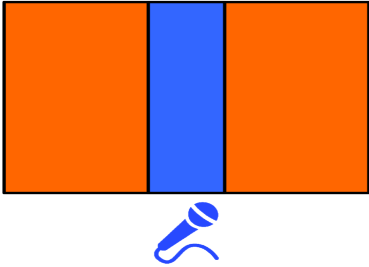
<ul style="list-style-type: none"> <li>• data source: camera images</li> <li>• data point format: local image patches centered around each pixel <ul style="list-style-type: none"> <li>▪ the subset of pixels used are those onto which Wobbler points are projected</li> </ul> </li> <li>• multiclass model: set of classes from proprioceptive predictions at current locale: <math>\{A, B, C\}</math></li> <li>• training data <ul style="list-style-type: none"> <li>▪ labels for training: proprioceptive predictions mapped onto Wobbler points and then onto image patches</li> <li>▪ location: current locale, inside path</li> </ul> </li> <li>• test data <ul style="list-style-type: none"> <li>▪ labels for evaluation: human (painted onto pixels in image)</li> <li>▪ location: current locale, outside path</li> </ul> </li> </ul>	<div style="text-align: center;"> <p>Current Locale</p>  </div>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------

Table 4.2: Self-Supervision Overview

Tables 4.1 and 4.2 together comprise the fully self-supervised framework. Each table separately has test data that is compared to human ground truth labels, and then confusion matrices and corresponding evaluation metrics are produced. (Refer to chapter 7 for these results.) The test data in table 4.1 is the training data in table 4.2, but is also evaluated separately. We can refer to each of these tables as a *type* of experiment for which evaluation metrics are produced.

In addition to these two types of experiments, we have two more types of experiments that we run as benchmarks, presented in tables 4.3 and 4.4. These benchmarks are used for relatively evaluating the accuracy of the self-supervised model. Specifically, they are evaluating the accuracy of the exteroceptive test predictions in the second step of the two-step framework, since this is the final output of the algorithm.

Table 4.3 is a floor benchmark. This is a traditional supervised vision scenario. In

this scenario, no proprioceptive data is used. This is how an exteroceptive visual classifier would perform without the aid of proprioceptive self-supervision. The data source is the set of camera images from the stopped waypoints. The data points are the local image patches centered around each projected Wobbler point. Note that the Wobbler points are not being used here for registration of the proprioceptive and exteroceptive data. They are still used to project the predictions onto the world map, and they are also used to threshold the data to under a certain range, discussed in chapter 6. The training data comes from the set of previous locales. This is the same set of locales we use in the proprioceptive experiment in table 4.1. It is a set of locales that covers the full set of terrain classes in the model. The test data is the current locale. The training data (in blue) and the test data (in orange) cover the entire map in each locale, not specific to inside or outside of the robot’s path. There is no concept of the robot’s path in these scenarios, because the robot is not interacting with the terrain to collect proprioceptive data. We still, however, use the robot’s path to collect image pairs from the stopped waypoints. These stopped waypoints are arbitrary here because they are not being used for registration of the proprioceptive and exteroceptive data. They are just being used so that our benchmark is consistent with the self-supervised experiment in table 4.2, and so that we have data from multiple vantage points. Having data from multiple vantage points allows the vision floor to produce better accuracy, thereby putting a harder constraint on the relative accuracy of our self-supervised algorithm. The training data is human-labeled (denoted with the blue human icon), because this is a purely supervised scenario. The test data is human-labeled only to provide ground truth for evaluating the accuracy of the classifier. The human labels for both the training and test data are painted onto the image plane with Gimp.



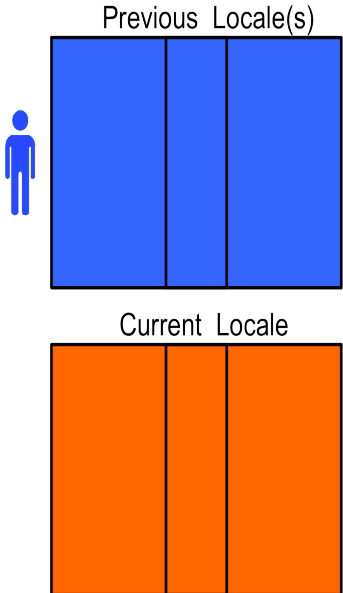
<ul style="list-style-type: none"> <li>• data source: camera images</li> <li>• data point format: local image patches centered around each pixel <ul style="list-style-type: none"> <li>▪ the subset of pixels used are those onto which Wobbler points are projected</li> </ul> </li> <li>• multiclass model: classes <math>\{A, B, C, D\}</math> (in reality, 7 classes)</li> <li>• training data <ul style="list-style-type: none"> <li>▪ labels for training: human (painted onto pixels in image)</li> <li>▪ location: previous locales, whole image</li> </ul> </li> <li>• test data <ul style="list-style-type: none"> <li>▪ labels for evaluation: human (painted onto pixels in image)</li> <li>▪ location: current locale, whole image</li> </ul> </li> </ul>	
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------

Table 4.3: Vision Floor Overview

Table 4.4 is a ceiling benchmark. The second step of the two-step self-supervised framework in figure 4.11 is what is presented in table 4.2. Now we replace the proprioceptively-predicted training labels with human labels, thereby cutting out the first step in the two-step framework. Similar to the floor experiment in table 4.3, this is a fully supervised vision scenario. But here, we train and test in the same locale. The training data is the data inside the robot’s path. The test data is the data outside the robot’s path. The purpose of this ceiling benchmark is to evaluate how well the exteroceptive classifier in the second step of the two-step self-supervised framework could do if the proprioceptive predictions were perfect. The performance of the ceiling will still be limited by the strength of the visual model, which is discussed in chapter 6. The performance is also limited by the ability of the training data inside the robot’s path to represent the terrain classes. Note that the multiclass model is trained on the subset of classes at this locale, classes  $\{A, B, C\}$ . This is the set of ground truth classes inside of the robot’s path. If there are classes  $\{D\}$  outside the robot’s path that do not show up inside the robot’s path, then this error will be propagated through, even in this ceiling benchmark scenario. The labels for the training data are human labeled. We have human labels from the Gimp painting that can be used, but we also have human labels as timestamps from the time series data. We use the time domain human labels in this step to label the training data. So the human label for each short time window is used, and then the same data registration process is used to associate each image patch data point with a label from a short time window. Besides being a ceiling benchmark, this experiment helped us to develop and debug the data registration process by controlling for the proprioception variable.

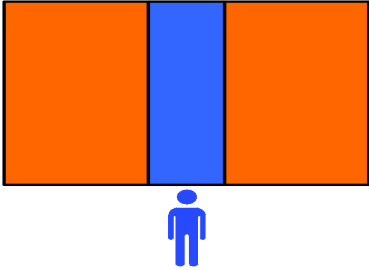
<ul style="list-style-type: none"> <li>• data source: camera images</li> <li>• data point format: local image patches centered around each pixel <ul style="list-style-type: none"> <li>▪ the subset of pixels used are those onto which Wobblers points are projected</li> </ul> </li> <li>• multiclass model: set of classes that actually show up in current locale: <math>\{A, B, C\}</math></li> <li>• training data <ul style="list-style-type: none"> <li>▪ labels for training: human (painted onto pixels in image)</li> <li>▪ location: current locale, inside path</li> </ul> </li> <li>• test data <ul style="list-style-type: none"> <li>▪ labels for evaluation: human (painted onto pixels in image)</li> <li>▪ location: current locale, outside path</li> </ul> </li> </ul>	<div style="text-align: center;"> <p>Current Locale</p>  </div>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------

Table 4.4: Vision Ceiling Overview

## 4.4 Thesis Problem Revisited

Now that we have introduced the main experiments in section 4.3, we can revisit the thesis problem statement from section 2.4, and discuss in more detail how we address the problem.

To increase the *semantic richness* of our world model, we will increase the number of terrain classes in our multiclass model. This will be achieved by using the best methods we can for our proprioceptive and exteroceptive modules, so that the two modules together can produce the best terrain predictions for the self-supervised framework. The proprioceptive module is discussed in chapter 5, and the exteroceptive module is discussed in chapter 6.

To decrease the *human effort* required, we will train a supervised proprioceptive model on a subset of locales. Our hypothesis is that a proprioceptive model will generalize to new locations better than an exteroceptive model. We will prove this by showing that supervised proprioception (table 4.1) out-performs supervised exteroception (table 4.3) when making terrain class predictions at a locale on which the classifiers have not been trained. We demonstrate this with our results in chapter 7. However, the proprioception can only make predictions on the subset of the locale with which it has interacted. The robot effort is still high with just this step.

To decrease the *robot effort* required, we will use self-supervision to decrease the amount of terrain interaction that is needed at each new locale. By using the proprioceptive model to train the exteroceptive model at each new locale, we leverage the ability of exteroception

to collect data on a larger part of the terrain map. We will prove that our algorithm is benefiting the robotic system by showing that self-supervision (table 4.2) out-performs supervised exteroception (table 4.3) when making terrain class predictions at a locale on which the classifiers have not been trained. We demonstrate this with our results in chapter 7. Furthermore, the self-supervision can make terrain predictions on the full set of data points at that locale, which allows the *robot effort* to be decreased for a given map size.



# Chapter 5

## Proprioception Module

In this chapter we present the proprioception module of the self-supervised framework. This is the first step of the two-step framework presented in chapter 4, figure 4.11. This module is supervised learning on the proprioceptive data for a multiclass model that produces one of seven terrain class predictions. After the model is trained on a set of locales, it predicts terrain types on a new locale (which we call the *current locale* in figure 4.11). This current locale will then be used for the self-supervised step, and the proprioceptive predictions will be used as self-supervised labels for the exteroceptive data. This chapter is focused on how these proprioceptive predictions are made.

In section 5.1, we review literature on how proprioception has been used in similar applications, which will motivate our feature extraction methods. In sections 5.2, 5.3 and 5.4, we go through the pipeline of turning data points into class predictions. This involves turning the signal into data points, extracting feature vectors from those data points, and then feeding those feature vectors into a classifier. In section 5.5, we present our early experiments and results on the Gator platform, using these techniques.

In section 5.6.1, we build off this work for the augmented proprioceptive sensor suite on the LAGR platform. We introduced this sensor suite in section 3.2. Now we describe the experiments performed on these sensors, and in particular, how we automated the sensor selection process. In section 5.6.2 we present our final results on the LAGR platform. These results are the predictions that will be used as training labels for the second step of the self-supervised framework.

### 5.1 Acoustics and Vibration Literature

Acoustics and vibration used for listening to the interaction of one object with another has primarily been explored in controlled laboratory environments where single impacts are recorded. Wu and Siegel placed microphones and accelerometers in the heads of hammers, which they tapped against sheet-like materials to determine their integrity [40]. Durst and Krotkov measured the sound of striking objects with canes and extracted spectral peaks as features to determine material type [12]. In later work, Krotkov et al. [21] and Amsellem and Soldea [2] performed similar experiments which also took the shape of the object into

account and then determined material type based on spectral energy decay. This worked well for those applications in which the signal source was a single clean impact of a cane striking against a resonant object in a laboratory setting with no background noise. In our scenarios, we sometimes have multiple impacts in succession (such as the side of the vehicle scraping against a rock), impacts with non-rigid bodies (such as splashing into water or compressing vegetation), or continuous interactions (such as the wheels of the vehicle rolling over ground). In these cases, spectral peaks cannot always be found. Furthermore, the vehicle drives past these interactions as it is creating them, so the microphones do not stay in range of the interactions long enough to listen to their decay.

Much of acoustic classification research has focused on characterizing time-varying structure, such as motor oscillations in an engine or recognizing words in speech. Wellman et al. classified engine sounds by looking for harmonics in motor oscillations. This involves having a periodic sound source, which will manifest patterns over time [39]. Hidden Markov Models (HMMs) and Markov Random Fields (MRFs) have been used to learn the correlation of speech over time [24], [31]. As mentioned previously, Krotkov et al. [21] and Amsellem and Soldea [2] extracted features from the spectral energy decay, where an evolution over time was one of the key factors. In our work we explicitly choose *not* to model time varying elements. There will definitely be dynamic changes in our data that could form word-like structures, such as the start, middle and end of scraping against a rock, but this would be learning the shape of a particular rock. As discussed earlier, we are explicitly trying *not* to use features such as shape, so that our classifiers can generalize to new environments. This also helps to simplify our models which is important since we will have a very limited number of data points.

Our work is most similar to applications where: 1) the signal is not constrained to a single impact, and 2) the signal is treated as a stationary process. This is demonstrated by Hoiem and Sukthankar [16] and Giannakopoulos et al. [14], where they looked at sounds from movies, such as dog barks, car horns, and violent content. Another example is from Martinson and Schultz, where a robot classified voice signatures to help identify people [23]. This is also similar to the vibration-based analysis presented in Brooks and Iagnemma [8], as discussed previously. In these cases, features were extracted by looking at short windows in time and treating each window as independent from other windows in the same sequence. For each window, temporal and spectral characteristics are considered. We take a similar approach in our work. This approach is discussed in the next two subsections. Subsection 5.2 discusses the process of turning signals into hand-labeled data points (short windows). Subsection 5.3 discusses the process of turning the data points into features.

## 5.2 Data Overview and Hand Labeling

In this section we discuss how the proprioceptive signal streams are turned into data points and how they are hand-labeled. The proprioceptive data is time-series data from the sound and vibration sensors. Providing human labels for the data means that we label the beginning and end of a terrain interaction sequence. By *terrain interaction sequence*,

we mean the occurrence of interacting with a particular terrain class. For example, a particular data collection trial might involve driving over grass, then driving over pavement, then driving over grass again, and then driving through a pile of bramble. So we have four sequences: a sequence of grass, then a sequence of pavement, then a second sequence of grass, and then a sequence of bramble. The trial will have a time series signal file for each proprioceptive sensor. Each signal file will have these four sequences contained within it, and we label the beginning and ending timestamps of each sequence. On the Gator, there is just one microphone and one vibration sensor. On the LAGR, there are many proprioceptive sensors. On the LAGR, these sensors have all been time-synchronized, so hand-labeling the timestamps in one signal file can automatically be used for any of the files. This allows hand-labeling to be much simpler, since we only have to do the labeling for one file. Furthermore, this allows us to swap back and forth between different sensor files, depending on what type of terrain we are trying to label. For instance, ground terrain types are more easily picked up from the sensors on the axle. Above-ground terrain types are more easily picked up from the sensors on the bumper.

We developed interactive software to aid in labeling our data, which incorporates a combination of camera images, time series plots, and audio feedback. This software allows us to swap out different proprioceptive sensor signal files, and then look at the time series plot for that file. We can then zoom in on an interesting area, such as a particular sequence or the boundary between one sequence and another. We might also listen to an audio playback of the signal, whether it is audio or vibration signal file, and then listen to only a zoomed-in portion of that file. The software then allows us to graphically click on the time series plots to mark the beginning and ending timestamps of each terrain interaction sequence. This software also allows us to refer to the image stream associated with a particular sequence. It is not always straightforward for a human listener to distinguish these events from just the sound information, so we used the camera images as a sanity check, or second reference point, to make sure we were listening to the correct part of the trial. On the Gator, the image stream was from a webcam. On the LAGR, the image stream was from the left and right Pixim cameras, which are subsequently used for the exteroceptive module of the self-supervised framework. The fact that our algorithms are successful only using proprioceptive data, while the human in this labeling process needs vision as well, speaks to the inherent power of proprioception as a computational tool.

For the ground classes, labeling was much easier, as these events covered very large spans of time. If the boundaries of the event are not labeled perfectly, it does not affect the majority of the data. One sequence might have a very long duration, and so many data points can subsequently have human labels by just marking the beginning and end of one sequence. For above-ground classes, we consider much smaller patches of terrain. This is also the case for ground classes such as puddles of water. The boundaries of these events are often the majority of the data, so the labeling of the boundaries must be handled with greater care. We must also label more of these sequences to acquire a significant amount of data. On the LAGR, we took the additional step of only looking at the subset of the signal file where the robot was moving at a maximum constant velocity of 0.5 m/s. Extracting this subset is an automated process, using the velocity output from the vehicle’s low-level control. The automated path behavior of the robot (i.e. stopping and starting at each

waypoint as discussed in section 3.4), along with the automated process of extracting out the maximum velocity sections of the signal, made labeling the ground data even easier. If a moving interval between two waypoints just covered a particular terrain, then we could automatically tag that interval with its terrain label, and the beginning and ending timestamps would be acquired automatically by using the velocity information. (This tagging would be done using information from the camera streams.) In these cases, we would only need to mark the beginning and ending timestamps within moving intervals that contain boundaries between one terrain type and another. We ignore these boundary moving intervals, since we have so much ground data anyway. We then only had to worry about marking the beginning and ending timestamps for above-ground terrain.

After we completed the labeling, we had sequences of time series data where each sequence represented an event of interacting with a specific class. We split these sequences up into short time windows. Each window is then a data point which is used for feature extraction and classification. We empirically chose a window size of 200 ms, with 50% overlap between successive windows. We found that 200 ms was the shortest window we could use that still gave decent results. If the window is too short, then the small dynamic elements in the data will generate too much variability from one window to the next. As discussed in section 5.1, we are purposely not characterizing these dynamic elements. Another problem is that the shorter the window, the less frequency resolution that can be captured when transforming the window into the frequency domain. On the other hand, shorter windows allow detection to happen more quickly, which is important for the eventual goal of online algorithms for autonomous robots. Furthermore, shorter windows also allow more resolution when looking at the boundary between one class and another. As mentioned above, for above-ground classes and certain ground classes that happen over a short period of time, the boundaries are almost all of the data; hence data point resolution near the boundary is extremely important.

### 5.3 Feature Extraction

In this section, we discuss how proprioceptive data points (short time windows) are transformed into feature vectors. We thoroughly experimented with a wide array of feature extraction techniques. These techniques came from prior work in acoustics and vibration, which we discuss in section 5.1. A few features are extracted from the time domain and the majority from the frequency domain. To obtain the frequency domain, we first smooth each time domain frame with a Hamming window, and then we apply a Fast Fourier Transform (FFT). We retain the spectral amplitude coefficients for further analysis, ignoring the phase. We normalize the signal first in the time domain and then in the frequency domain. In the time domain, we normalize the amplitude of the signal to range between  $[-1,1]$ , and then in the frequency domain we normalize the spectral distribution to sum to 1. Normalization makes our algorithms blind to volume, which prevents the classifiers from overfitting to factors such as the capture level of the microphone, the specific microphone being used, the distance from the microphone to the sound source, and certain variations across events from the same class.



We extract three features from the time domain. The first is the zero crossing rate (ZCR), which is the number of times per second that the signal crosses the zero axis. The second is the short time energy (STE), which is the sum of the squares of the amplitudes. The third is the energy entropy, which is a measure of the abrupt changes in energy present in the signal. To calculate the entropy, the frame is subdivided into  $k$  subframes, with  $k = 10$ , chosen experimentally. The energy of each subframe is calculated as the short time energy (described above), and then normalized by the energy of the whole frame. The entropy is then calculated by  $E = -\sum_{i=0}^{K-1} \sigma^2 \cdot \log_2(\sigma^2)$  where  $\sigma$  is the normalized energy of a subframe.

We extract many features from the frequency domain. The most direct is treating each amplitude coefficient in the raw spectrum as a dimension in a feature vector. In converting our proprioceptive sensor signals from analog to digital, we worked with a variety of sampling rates, ranging between 12.5 kHz and 51.2 kHz, depending on the sensor and data acquisition board being used. Lets consider a typical microphone that has a sampling rate of 44.1 kHz (CD quality). For this sampling rate, using the entire spectrum is computationally overwhelming, so we considered how we might use the parts of the spectrum that contain the most information. Figure 5.1 shows spectrograms for terrain classes used in early Gator experiments. This data is from the front snowball microphone. The x axis is time. The labeled terrain interaction sequences for each class are concatenated together in succession over time along the x axis. The colors show the percentage of spectral magnitude at each frequency, with red being the highest and blue being the lowest. One can see from this figure that the majority of the spectral power is in the lowest end of the spectrum. This lead us to experiment with only using the lower part of the spectrum, varying the truncation point. Using a truncation point of 0.4 kHz allowed us to work with a manageable 200-d feature vector. (0.4 kHz is 400 Hz. The Nyquist frequency is then 200 Hz, leading to 200 spectral amplitude coefficients in our feature vector.) The rationale for truncating to the lower end of the spectrum comes from the fact that the signal-to-noise ratio (SNR) is lower in regions of the spectrum where there is less power. However, these low SNR regions could contain some information, and this information might be very pertinent to distinguishing between different classes, so we do not necessarily want to ignore these regions altogether. If the information about the terrain is inherently contained in these high-frequency bands, then even if the SNR ratio is low, we still want to consider that data. Binning the spectrum along the spectral amplitude coefficients allows the entire spectrum to be captured while still reducing the dimensionality. This is discussed in Hoiem and Sukthankar [16] and Peeters [28]. We experiment with bins that are spread log-uniformly along the frequency scale, which focuses on the lower frequencies. This technique allows us to reduce the feature dimensionality and focus on the lower frequency bands with high SNR, while not altogether ignoring the higher frequency bands with low SNR.

We also look at parametric techniques for characterizing the frequency distribution. We compute the moments of the distribution: the centroid, standard deviation, skewness, and kurtosis. (See Wellman et. al. [39] for equations.) We also compute the spectral rolloff, which is the frequency value under which a certain percentage of the total power lies. We use a value of 80%, determined empirically. We also compute the spectral flux, which

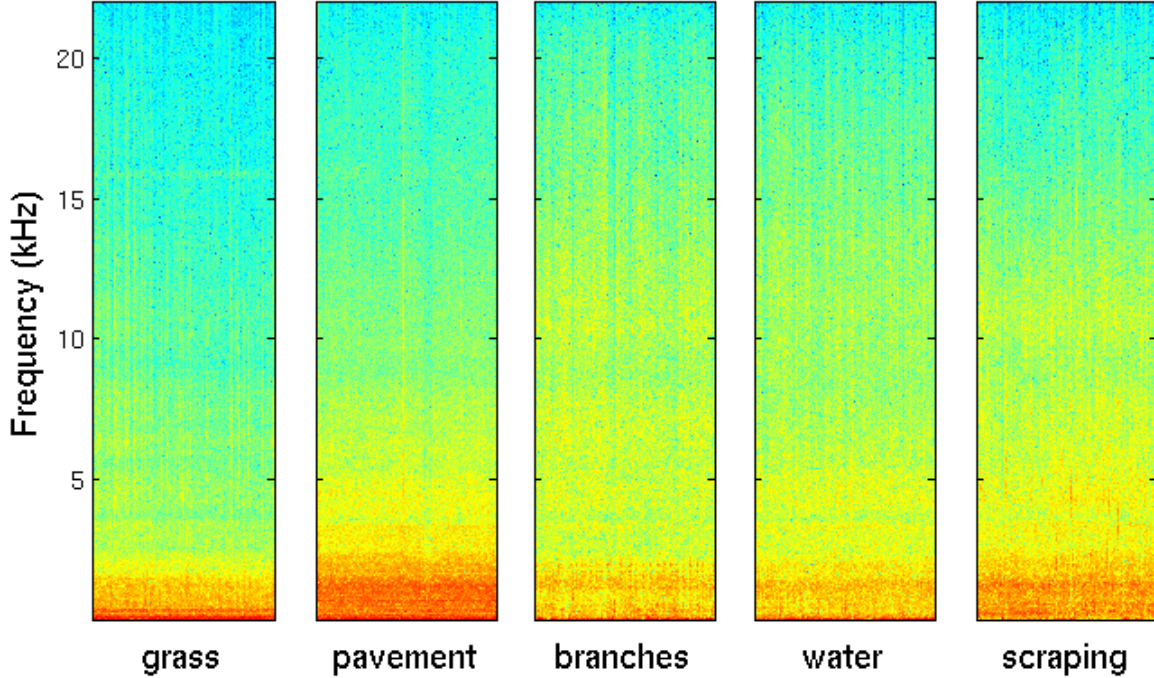


Figure 5.1: Spectrograms for each class. The x axis is time. The labeled terrain interaction sequences for each class are concatenated together in succession over time along the x axis. The colors show the percentage of spectral magnitude at each frequency, with red being the highest and blue being the lowest.

measures the rate of energy change. The flux is computed by taking the difference between the coefficients of consecutive frames and then summing and squaring the differences.

The various scalar quantities presented above are combined into feature vectors. Giannakopoulos et. al. [14] looked at six of these scalars in particular: ZCR, STE, energy entropy, spectral centroid, spectral rolloff, and spectral flux. In following these methods, we combine these scalar values into a 6D feature vector. Wellman et. al. [39] looks specifically at the distribution moments of the spectrum, and in following these methods, we create a 4D feature vector consisting of the spectral centroid, standard deviation, skewness and kurtosis.

To summarize, we take each 200 ms frame as a data point, and process it in different ways to form various feature vectors. We delineate these vectors in the following list, which are each used in separate trials as inputs into our classification algorithms.

- Raw spectral coefficients, truncated at 0.4kHz
- Bins of spectral coefficients, with logarithmic binning.
- 6D vector of temporal and spectral characteristics, derived from Giannakopoulos et. al. [14], which we will refer to as the “gianna” features.
- 4D vector of spectral moments (“shape” features).

- 9D vector which combines the 6D and 4D vectors above. We call this the “gianna and shape” vector. (Note that this is not 10D because one of the dimensions is present in both original vectors.)

## 5.4 Classification

For classification, we use Support Vector Machines that are implemented in the SVM-Light library (Joachims [18]). Since this is a new form of mobile robot proprioception, we do not have previous datasets to work from. The datasets we collected are small in comparison to the online data banks from which many machine learning applications benefit. SVMs are a good choice for handling small datasets because they are less prone to overfitting than other methods. SVMs also have a sparse solution, allowing for predictions to be very fast once the classifier is trained. This is important for online applications such as mobile robotics. (See Burges [9] for more discussion on these motivations.)

We experiment with both linear and radial basis function (RBF) kernels. RBF kernels sometimes perform better than linear kernels because they can handle non-linear separations, and they sometimes perform better than polynomial kernels because they have fewer parameters to tune and less numerical instability. Linear kernels, however, are sometimes better for high dimensional feature spaces.

We build a multiclass classifier with the standard *one-vs-one* approach: we train binary SVMs between each pair of classes, and then each binary classifier is a node in a Decision Directed Acyclic Graph (DDAG), as described in Platt et. al. [29]. A nice analogy for the DDAG graph is the following: imagine you have a game with multiple players. In each round of the game, two of the players compete. (This competition between two players represents a binary node.) Whichever player loses the round gets kicked out of the game. The winner then competes against another player, and the process repeats until there is only one player left in the game.

The best way to use SVMs for multiclass classification is an open question. We choose the *one-vs-one* approach (as opposed to using a single multiclass SVM) so that we can decompose the problem into a series of simpler binary problems. It is also faster in training time than a single multiclass SVM, which allows us to experiment with many different hyperparameters and feature combinations. *One-vs-rest* is another decomposition technique, but this approach involves another layer of parameter fitting to resolve the differing scales between binary nodes. With that being said, decomposition methods such as *one-vs-one* and *one-vs-rest* are both popular techniques, and standard SVM libraries such as LIBSVM and Weka use these techniques respectively for their multiclass implementations.

When training each binary *one-vs-one* node, we experiment with linear and RBF kernels, as mentioned above. For these kernels, the hyperparameters must be tuned. The linear kernel has one hyperparameter: slack. The RBF kernel has this slack parameter, as well as an additional sigma parameter for its Gaussian. The linear SVM is a linear hyperplane that separates two classes in a dataset. The *support vectors* are the data points that determine this hyperplane. They are the points closest to the hyperplane, and the SVM’s optimization process tries to maximize the distance from these points. The slack

parameter defines how many data points can fall on the wrong side of the hyperplane. This helps in avoiding overfitting. It is a trade-off: the more slack there is, the less overfitting there will be, but the more error there will be in data points that fall very close to the hyperplane. With the RBF kernel, the hyperplane is no longer linear, or flat. Imagining a 2d feature space, one might think of the RBF hyperplane as interconnecting ellipses that surround clusters of data. The sigma parameter is like the sigma value in a 2d elliptical Gaussian around each data cluster. Again, there is trade-off: the larger the sigma value, the larger the ellipses will be, and the fewer clusters there will be to represent the data. This will lead to less overfitting, but more errors in the data points that fall close to the boundaries.

In order to tune the hyperparameters, we perform an automated cross-validation process. For one choice of hyperparameters, a binary node can be trained. For each binary node, we loop through different combinations of hyperparameter values. (For a linear SVM, this is just a choice of slack value. For an RBF SVM, this is a combination of a slack value and a sigma value.) For each combination, we train an SVM, and compute the cross-validation accuracy on that model. (Our metric for accuracy here is the average recall rate of the two classes in the binary node.) The combination with the highest accuracy is then used for our final model.

As a benchmark comparison, we also use k-nearest neighbor (k-NN) with inverse distance weighting. We use an l2 distance metric. We use  $k = 10$ , which was determined empirically. This means that for each binary decision in the DDAG tree, we replace the learned SVM classifier with a k-NN classifier.

## 5.5 Gator Experiments and Results

This section presents early experiments and results conducted on the Gator platform. We first present our results using just acoustic data, as published in our ICRA paper [22]. We explain in detail how we conducted these trials. We then present further results using acoustics *and* vibration data, which extends the techniques used in the acoustics trials.

Table 3.1 from section 3.1 showed the number of data points for each terrain class. Here in table 5.1, we break these up into training and test sets. Since we had multiple locations for each class, we never had the same location in both the training and test sets for any given class. Our results therefore prove the ability of our models to generalize to new locations. Note that there was much less data for the last two classes because these interactions involved short events, whereas for the first four classes we had more continuous stretches of terrain with which to interact. To preserve symmetry, we trained on the same number of data points for each class. Since 89 was the minimum number of training points, we randomly chose 89 points from the larger data sets to use for the training. We did not limit the number of data points in the test data.

We used learned models on separate trials for different feature and classifier combinations. We experimented with the five feature sets as listed in bullet points at the end of section 5.3 and three classifier variants: SVM with an RBF kernel, SVM with a linear kernel, and k-NN. For each combination, we trained the binary classifiers separately, where

Class	Training	Test
Grass	1364	1373
Pavement	1856	1377
Gravel	3986	4347
Water	1787	3384
Hard objects	203	65
Wheels spinning	89	244

Table 5.1: Number of training and test data points used in Gator acoustics experiments

each binary classifier is a node in the *one-v-one* graph. (For the k-NN classifier, there was no training involved since we chose  $k$  empirically.)

For each binary node, we performed the following steps. First, we zero mean shifted and whitened the training data. These transformations were stored as part of the model for that node and then used as well on the test data. We then used *leave-one-out* cross validation on the training data to tune the SVM parameters.

Once we completed the training on the binary nodes, we fed the test data into the *one-v-one* graph, and we determined accuracies by comparing the true labels to the predictions that are output from the graph. Figure 5.2 shows accuracies for each of the feature/classifier combinations. The data points here are just coming from the front microphone. Full results would show a 6x6 confusion matrix for each trial. For compactness, we condensed each confusion matrix into one number, the *balanced accuracy*, which is the mean of the diagonals in the matrix, or rather, the mean true positive rate for each class.

Note that for a binary classifier, 50% accuracy would signify no information. Across six classes, this number would be 17%. The lower the performance of a trial, the more that trial was overfitting to the training data. The worst results were given by the raw FFT feature vector. Since this vector has the highest dimension, it makes sense that it would overfit. The *log bins* vector overfitted with the SVM RBF kernel but then performed very well with the SVM linear kernel. This is plausible considering this vector still has a relatively high dimension; since the SVM RBF kernel adds another parameter to the classifier, this increases the chances of overfitting. The lower dimensional feature sets performed well across both SVM variants.

The two trials that performed the best are the *gianna and shape* feature set with an RBF kernel and the *log bins* feature set with a linear kernel. We used these two combinations to run further trials. In these new trials, we experimented with how to combine the data from both microphones. The easiest method is to always use the back microphone for the *hitting hard objects* class and use the front microphone for all other classes. This means that for both the training and test sets, we explicitly select which microphone to use. This is what we did for the results presented above in figure 5.2. One could argue that this is giving the test data too much information, so we also ran trials where we used both microphones for all classes. This means that for each short time window in our data set, we really have two short time windows, one from each microphone. Figure 5.3 shows our experiments for different ways of combining these two data points. The left-most group,

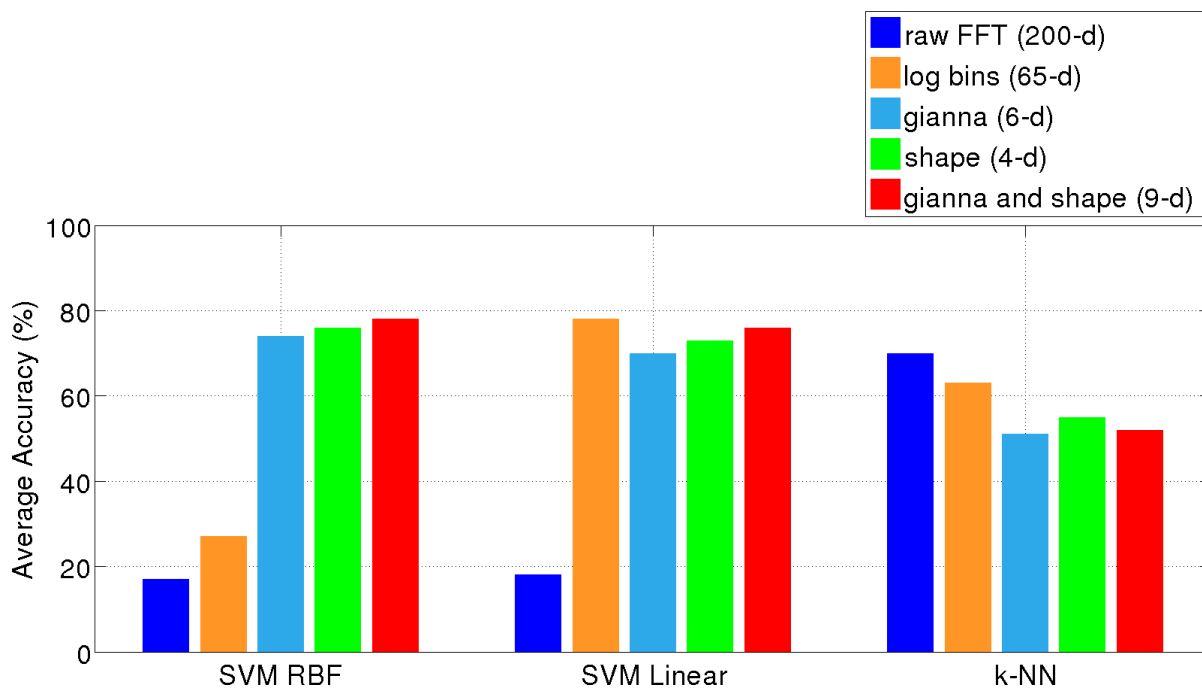


Figure 5.2: Bar chart of balanced accuracies from acoustics experiments for the different feature and classifier combinations using just the front microphone. Each main group along the x-axis is a classifier choice. Within each group, the legend specifies the various feature sets used. The dimension of each feature choice is given in parentheses.

*choosing mic*, is the original method of choosing which microphone to use for which class, and so these two bars are the same as in figure 5.2. The next three groups are different ways of blindly taking data from both microphones. *Doubling data* means that the windows from each microphone were treated as separate data points. *Doubling dimension* means that the feature vectors from each microphone were concatenated together. *Averaging* means that each dimension of the two feature vectors were averaged. Again, each bar is the balanced accuracy for a particular trial. The *averaging* method performed the best. *Doubling dimension* also performed very well for the *gianna and shape* vector, but the *log bins* vector suffered. Since the *log bins* vector already has a very high dimension, it makes sense that doubling the dimension would cause problems. Treating the microphones as separate data points (*doubling data*) did not perform very well because the data coming from each microphone is inherently different. The distance from a particular sound source should not in itself have much of an effect since we normalize the volume, but the ratio of source sound to background sound would be quite different for the two microphones.

We take the best result from above that uses both microphones (*averaging* with *gianna and shape*), and we further improve the results by applying a mode filter to smooth out noise. Each short time window exists within a time sequence, and we slide a larger window across each sequence that is five times the size of each short time window data point. We tally the votes for each data point's prediction in this larger window, and use this vote to relabel the data point in the center of the window. Note that even though we are using

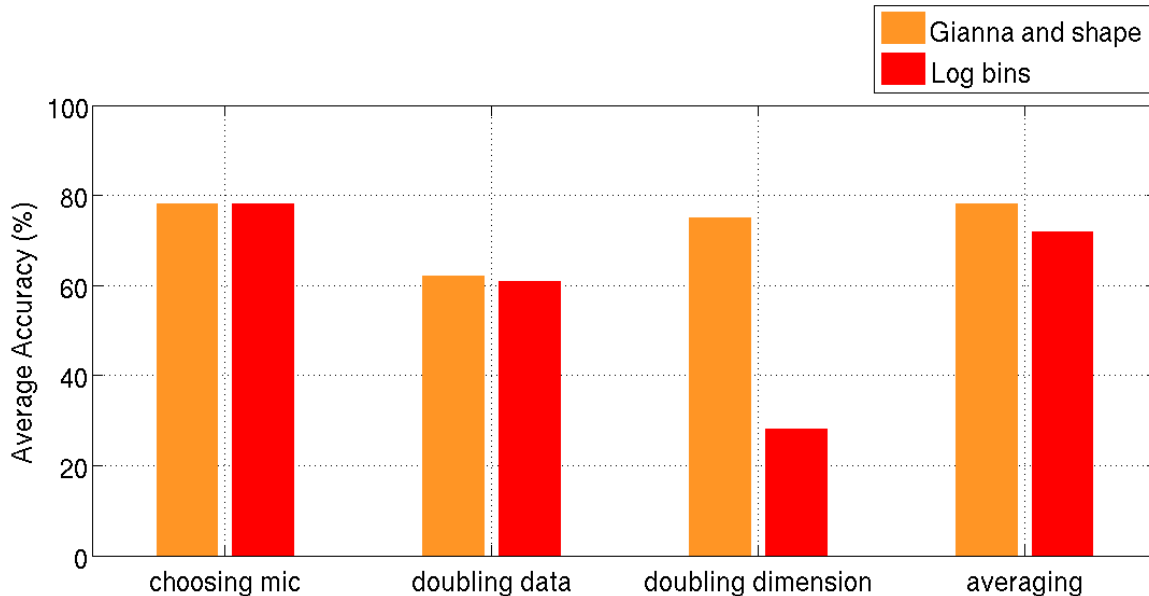


Figure 5.3: Bar chart of the balanced accuracies from acoustics experiments for different ways of combining data from the two microphones. The two feature vectors used are depicted in orange and red. Each main group along the x-axis specifies a different way of combining the data.

the time dimension to help with noise smoothing, we are still ignoring this dimension in the training process so that the time-varying structure is not modeled. This smoothing step increases the balanced accuracy from 78% to 92%. We show the normalized confusion matrix for this best result after smoothing in table 5.2 below. The class that performs the worst is the pavement class, with a true positive rate of 70%. This class exhibits the most confusion with the other benign road classes, the grass and the gravel.

		Actual Label					
		Grass	Pavement	Gravel	Water	Hard obstacle	Wheels spinning
Predicted	Grass	<b>98</b>	9	1	1	0	0
	Pavement	0	<b>70</b>	1	0	0	0
	Gravel	1	19	<b>97</b>	2	2	1
	Water	0	1	0	<b>96</b>	0	2
	Hard obstacle	0	0	0	1	<b>98</b>	5
	Wheels spinning	1	1	1	0	0	<b>92</b>

Table 5.2: Normalized confusion matrix from the acoustics trial with the best average accuracy of 92%

We then incorporated vibration data into our experiments. These trials improved classification accuracies, suggesting that the vibration data complements the acoustic data.

Since the vibration sensor is on the front wheel axle, we discarded the *hitting hard objects* class, since this interaction was happening near the rear of the vehicle. In turn, we only used data from the front microphone. We also discarded the *wheels spinning* class because we did not have enough vibration data for these events. The training and test data sets for these trials were slightly different, simply because we mounted the vibration sensor later, so some of our previous acoustic data collections could not be used. Unfortunately, this leaves us with no *above ground* classes for these experiments, but we did still have the *water* class, which is not addressed by prior work in vibration classification. Even though we had no above-ground classes, these experiments are still useful in showing that at least for ground classes, vibration data can improve the accuracies over using just acoustic data alone.

For these trials, we used 200 ms windows for both the acoustic and vibration data. We turned each window separately into a *gianna and shape* vector. We experiment with using these vectors separately, as well as concatenating them into one longer vector. By concatenation, we mean doubling the dimension, as shown as a technique in figure 5.3. We could have also tried averaging with the *log bins* feature vector, but figure 5.3 showed that the accuracy is comparable between these two methods, so we went with the lower-dimensional choice. (Doubling the dimension on the 9-d *gianna and shape* vector is still a lower dimension than the 65-d *log bins* feature vector.) The vectors are fed into an SVM multiclass classifier with an RBF kernel. Again, the multiclass classifier is built from binary SVM nodes with a DDAG graph. We used separate data locations for training and testing.

Table 5.3 shows the results for these trials. Each row is a separate experiment, for a particular choice of sensors. The numbers in each cell are the true positive rates for each class. (Each row is the main diagonal of the confusion matrix for that experiment.) The first row just uses features from the microphone, the second row just uses features from the vibration sensor, and the third row combines both features by concatenating the feature vectors from each of the sensors. For every terrain class type, the results are improved by combining the sensors, as can be seen by the higher accuracies in the third row. In comparing the first two rows, we can see that the microphone always outperforms the vibration sensor. However, even though the vibration sensor does not perform well on its own, the combination of the two sensors still outperforms the microphone. This demonstrates that the vibration sensor is providing inherently different information than the microphone, so the combination of the two sensors provides more information than either one alone. We did not apply a final mode filter to these results, although this final step would improve the accuracies for any of the three rows. The goal here was to look at the unsmoothed predictions and analyze if the vibration data could complement the acoustic data.

In the next section we describe our work on the LAGR platform, but before moving on, we mention briefly here our work on a robotic manipulator. We implemented our supervised proprioception on this platform in order to test our approach for generality. In Bagnell et al. [6], we used a microphone mounted on the DARPA Autonomous Robotic Manipulation platform to classify the speed of a drill. A robotic arm autonomously picked up and turned on a drill, and our acoustics classifier determined if the robot was successful at turning on the drill to a full speed. Our classifier was able to distinguish between three



		Classes			
		Grass	Pavement	Gravel	Water
Sensors	Microphone	91	64	94	86
	Vibration	76	60	58	66
	Combined	93	73	96	91

Table 5.3: Classification accuracies for different terrain types, using microphone *and* vibration signals. The first two rows show accuracies when using these sensors separately. The last row shows improved results when combining the features from both sensors.

different speeds (low, medium, and high) with 100 percent accuracy in both recall and precision. We again used the *gianna and shape* feature vector with binary SVM’s in a *one-vs-one* graph. The accuracy was particularly high because the data in an indoor robotic manipulation lab setting is very clean compared to the data we work with in our outdoor off-road mobile robotics settings.

## 5.6 LAGR Experiments and Results

On the LAGR platform, we ran similar experiments to those that we did on the Gator platform, and then we extended upon these experiments with a learned sensor selection technique. This learned sensor selection allows us to take advantage of the augmented proprioception sensor suite that we have on the LAGR platform. In section 5.6.1, we describe the learned sensor selection process. In section 5.6.2, we present our final test results for the proprioceptive model. This final model will be used as the proprioception module in the first step of the two-step self-supervised framework.

### 5.6.1 Learned Sensor Selection

On the LAGR platform, we have many more proprioceptive sensors than we did on the Gator platform. We introduced these sensors in chapter 3. Table 3.4 listed the six main types of sound and vibration sensors used, and figures 3.6 and 3.7 showed these different sensors mounted at various locations on the vehicle. Even though there are only six types of sensors, some of them come in different sensitivities, and some varieties are mounted at multiple locations on the vehicle. Also, some of the sensors have multiple signal outputs for different axes.

For ground terrain classes, we are most interested in looking at the vertical axis (the vehicle frame’s z-axis). Our reasoning here is that the material of the ground will impact the wheels in mostly a normal direction to the surface of the wheel with the ground. This will then propagate through the axle of the vehicle, where it is measured by the axle sensors. It could also propagate through the chassis of the vehicle, and so we measure this with the bumper sensors as well. For above-ground terrain classes, we are most interested in looking at the along-track axis on the bumper (the vehicle frame’s x-axis). When the bumper drives into an above-ground terrain type, it will be mostly impacted along the

along-track axis. Sometimes we include all three x-, y- and z- axes, but often we leave some of them out to avoid unnecessary analog to digital processing. Taking into account the different varieties of sensors, the different placements on the vehicle, and the different axis outputs, we have a total of 19 sensor signals, listed below in table 5.4.

Sensor Signal
snowball
imu-xAccel
imu-yAccel
imu-zAccel
vt500-axle
adxl-axle-forward
adxl-axle-up
contact-axle-up
iepe10g-axle-up
iepe50g-axle-up
vt500-bumper
adxl-bumper-forward
adxl-bumper-up
contact-bumper-forward
contact-bumper-up
iepe10g-bumper-forward
iepe10g-bumper-up
iepe50g-bumper-forward
iepe50g-bumper-up

Table 5.4: List of 19 Proprioceptive Sensor Signals

We describe here what we mean by each signal name listed in table 5.4. The first entry is the *snowball* sensor. There is only one of these sensors mounted centrally on the vehicle. Since it is an omnidirectional air microphone, we do not consider different axes for it. The fifth and eleventh entries are *vt500-axle* and *vt5000-bumper*. The first part of the name, *vt500*, denotes the type of sensor. This is the Voice Technologies 500 electret waterproof microphone. Since it is an electret air microphone, we again do not have axes to consider. We have two of these sensors mounted: one on the axle and one on the bumper. The second part of the name, *axle* or *bumper*, denotes which of the two mounted sensors we are considering. The second, third and fourth entries are *imu-xAccel*, *imu-yAccel* and *imu-zAccel*. These are for the accelerometer inside the IMU. Since it is a 3-axis accelerometer, it has three separate signals to consider for the x-, y-, and z- axes. The first part of the name, *imu*, denotes the output is coming from the imu. The second part of the name, *xAccel* or *yAccel* or *zAccel*, denotes which acceleration axis is being considered. Even though we are mostly concerned with the y- and z- axes, we include all three since we automatically get them from the low level computer’s output. The sixth, seventh, twelfth and thirteenth entries are *adxl-axle-forward*, *adxl-axle-up*, *adxl-*

*bumper-forward* and *axle-bumper-up*. These are for the *adxl*, which is short for the Analog Devices 335 accelerometer. This is the accelerometer ubiquitous in most IMU's, and we measure it here outside of the IMU so that it is unfiltered, and mounted closer to the terrain-interaction points of contact on the vehicle (the bumper and axle). Again, the accelerometer has x-, y-, and z- axes. We consider two out of the three axes: the along-track axis and vertical axis. We have two of these sensors mounted: one on the axle and one on the bumper. The first part of the name, *adxl*, denotes the type of sensor. The second part of the name, *axle* or *bumper*, denotes which of the two mounted sensors we are considering. The third part of the name, *forward* or *up*, denotes which axis on a particular sensor we are considering. (*Forward* is along-track and *up* is vertical.) We are more concerned with the along-track axis on the bumper and vertical axis on the axle, but we include both axes at both locations, since we have them available. We could have included all three axes at both locations, but there were not enough inputs on the Labjack data acquisition board that we were using for these sensors. The eighth, fourteenth and fifteenth entries are *contact-axle-up*, *contact-bumper-forward* and *contact-bumper-up*. The first part of the name, *contact*, denotes the type of sensor. This is the contact microphone, which is classically used as a guitar pickup, suctioned onto the body of a guitar. Each contact microphone has one output. There is no formal axis for such a microphone, but we treat the axis as normal to the surface it is suctioned to. We have one of these sensors mounted on the axle, suctioned to a part of the axle that is parallel to the ground plane, so that it is normal to the vertical axis. (This is the *contact-axle-up*.) We have two mounted to the bumper. One is suctioned to the inside surface of the bumper such that its normal axis is along the vertical axis, and one is suctioned to another inside surface of the bumper such that its normal axis is along the along-track axis. (These are the *contact-bumper-up* and *contact-bumper-forward*, respectively.) The ninth, tenth, sixteenth, seventeenth, eighteenth and nineteenth entries are the *iepe10g-axle-up*, *iepe50g-axle-up*, *iepe10g-bumper-forward*, *iepe10g-bumper-up*, *iepe50g-bumper-forward* and *iepe50g-bumper-up*. The first part of the name, *iepe10g* or *iepe50g*, denotes the type of sensor and the sensitivity. These are the Integrated Electronics Piezo-Electric (IEPE) Measurement Specialties vibration sensors, in the 10g and 50g sensitivities, respectively. The second part of the name, *axle* or *bumper*, denotes where each sensor is mounted. The third part of the name, *forward* or *up*, denotes which axis is being measured. These are single axis sensors, and again we mount them to surfaces such that we are measuring the axis normal to that surface.

On the Gator, we had three sensor signals: one front air microphone, one back air microphone, and one single-axis vibration sensor. Section 5.5 explored different ways that we manually combined the data from these signals. We found that concatenating *gianna and shape* feature vectors from two sensor signals was the best method. Now we have 19 sensor signals with different potential benefits (sensing element, sensitivity, bandwidth, vehicle location, axis measurement). We could look at  $\binom{19}{k}$  different combinations of these signals, with  $1 \leq k \leq 19$ . Using the 9-d *gianna and shape* vector, when we concatenate features together, the vector dimension will increase to  $9*k$ . In order to limit the dimension to something reasonable, we limit  $1 \leq k \leq 2$ . For  $k = 1$ , we are considering each signal alone. For  $k = 2$ , we are looking at each combination of two sensor signals.  $\binom{19}{1} + \binom{19}{2} = 190$  combinations. For each of these combinations, we also look at both 0.2 second and 0.4

second short time windows, so now we have 380 different data sources. This is too much for us to consider manually as we did in the Gator experiments. So now we automate this process by looping each combination through our classifier, and then choosing the combination with the best accuracy. In this way, we are learning the sensor selection.

Table 5.5 presents the locales that we use for our training and validation data sets. By *validation* data set, we mean the *test* data set for this learned sensor selection process. We do not call it *test* data in order to distinguish it from the test data that we use in our final results for the full self-supervised framework. This validation data is being used to learn (or train) the choice of sensors. If we were to use the final test data for this step, then we would be using our test data for tuning our model, which could result in overfitting to our test data. We are training a seven-class multiclass model. The semantic terrain classes in this model are: grass, grass-leaves, pavement, vegSoft, bramble, bush, and treeBig. (*vegSoft* is an abbreviation for soft vegetation, and *treeBig* is an abbreviation for a tree with a large trunk that doesn't have any compliance.) Each locale only has some subset of the seven classes. The training set must encompass enough locales to span the seven terrain classes. We always train and validate on different locales. The validation set only has two out of the seven classes, *bramble* and *grass*. Ideally we would have enough locales in our validation set to span all seven classes, but this was not possible with the limit on the amount of data that we had time to label. Nevertheless, we can evaluate accuracies for these two classes. Note that these classes are still being predicted from the full seven-class classifier. Table 3.6 from section 3.4 presented the number of proprioceptive human-labeled data points for each class in each locale. Working from tables 3.6 and 5.5, we present in table 5.6 the number of training and validation data points we have for each class. Note that the term *NA* is used to denote classes for which we have no data points in our validation set.

	<b>Locale</b>	<b>Classes in that locale</b>
<b>training</b>	sea2-11	grass, vegSoft
	wood1-7	bramble, grass
	bush4-1	bush, grass
	tree2-1	treeBig, grass-leaves, pavement
<b>validation</b>	veg2-1	bramble, grass

Table 5.5: Training and Validation Locales for Proprioceptive Sensor Selection on LAGR platform

In table 5.6, the first three rows present our ground terrain classes: *grass*, *grass-leaves* and *pavement*. The next four rows present our above-ground terrain classes: *vegSoft*, *bramble*, *bush* and *treeBig*. They are listed in descending order of compliance. This is also correlated to the descending order in the number of data points. Soft vegetation is a completely compliant terrain type, and the robot can drive through it for as long as it wants. Bramble is somewhat compliant. The robot can drive through sparse bramble, and will eventually get stuck as the bramble becomes denser. Bushes are not so compliant. The robot will be stopped by the bush, but the bush will comply a little, allowing the robot

Class	Training	Validation
grass	619	127
grass-leaves	102	NA
pavement	24	NA
vegSoft	132	NA
bramble	94	42
bush	11	NA
treeBig	2	NA

Table 5.6: Number of Training and Validation Data Points for Proprioceptive Sensor Selection

to move a few inches into it. During these few inches, some time will transpire, creating a time sequence that can be split up into eleven 200ms short time window data points. Note that there is only one locale, bush4-1, where bush data is collected. Therefore we only have one interaction sequence that we are working with. Trees are not compliant at all. The robot will hit the tree and immediately come to a stop. This will look like a short spike in the time series data. There is only one locale, tree2-1, where tree data is collected. Therefore we only have one of these short impacts. Each short time window is longer than the impact itself. We have two data points instead of one because the short time windows overlap by 50%.

We presented the number of data points for our early Gator experiments in section 5.5, table 5.1. Even for our *hard objects* terrain class, we had 203 training data points and 65 test data points. This is because we had a human driving the vehicle for these data sets, so the human driver would bump into the hard objects over and over again. Now on the LAGR, the data collection is automated. This is partly because remote-controlling the LAGR is not easy, but also because we wanted automated control in order to more easily register the proprioceptive and exteroceptive data. The paths we took were slow and precise, partly due to the limitations in our localization estimates. We discussed these considerations in sections 3.4 and 4.1. As a result, we have much fewer impacts with non-compliant terrain. This is also partly due to constraints on the amount of time we have had to hand-label data.

Because we have so few data points for our non-compliant above-ground terrain types, we choose the data for our training set a little differently than before. For the Gator experiments, 89 was the minimum number of training points for any class. So we used 89 points from each class. Now on the LAGR, our minimum number of training points is 2. This is an unacceptably low number to use for training. So instead, we limit by the smallest ground class. In this case, this would be 24 data points, limited by the pavement class. (Refer to table 5.6.) For the classes that have greater than 24 data points, we randomly choose 24 out of the larger set. For the classes that have less than 24 data points, we repeat the data points to create 24 data points. In repeating the data points, one method is to add a little noise to the data to make them slightly different, but we did not find that this improved our results. Having repeated data points is accepted by the SVM-light software, and the classifier performs well. One theory for why this works is

because the feature representation of these non-compliant classes is far enough away from the other data points in the feature space such that data repetition does not break the algorithm. When we repeat data points for ground classes, the classifier performs poorly. For instance, if we were to use the maximum number of data points, 619, from the grass class, as the number of data points per class, and then repeat data points for every other class, the classifier performs poorly.

Now that we have presented the number of data points for these experiments, we describe the sensor selection process. Our loop has 380 iterations (as described above in this section for sensor combinations and window size choices). Each iteration defines the data points being fed into the model. The data points are consecutive overlapping short time windows from the time series proprioceptive signals. These data points are turned into *gianna and shape* feature vectors. These are 9-d vectors for the  $\binom{19}{1}$  iterations, and 18-d vectors for the  $\binom{19}{2}$  iterations. The vectors are fed into a multiclass classifier. As before, the multiclass classifier is built from binary SVM nodes with a *one-vs-one* Decision Directed Acyclic Graph (DDAG). (Refer to section 5.4.) A Radial Basis Function (RBF) kernel is used, since our Gator experiments showed that RBF kernels outperform linear kernels for low-dimensional feature vectors. The training data is zero-mean shifted and whitened before the SVM is trained. These transformations are stored as part of the model for that node, and then used as well on the test data. And again as before, the sigma and slack hyperparameters for the RBF SVM binary nodes are automatically tuned through an iterative cross-validation process. (Again refer to section 5.4.) In this inner cross-validation loop, we hold out half of the training data for each iteration, and test on the other half.

Once the seven-class multiclass model has been trained on our training data, we generate a confusion matrix for our validation data. Table 5.7 shows the confusion matrix for one of the 380 iterations. This iteration is a  $\binom{19}{1}$  iteration for the snowball sensor, with a 0.2sec window. The columns in the confusion matrix are the ground truth human labels. The rows are the classifier predictions. Only the *grass* and *bramble* classes exist in the validation data, so these are the only non-zero columns in the confusion matrix. Note that we are not smoothing the predictions with a mode filter (as described in section 5.5). We will do this smoothing at the very end of the proprioception module in our self-supervised framework.

		Actual Label						
		grass	grass-leaves	pavement	vegSoft	bramble	bush	treeBig
Predicted	grass	46	0	0	0	0	0	0
	grass-leaves	16	0	0	0	0	0	0
	pavement	25	0	0	0	0	0	0
	vegSoft	29	0	0	0	15	0	0
	bramble	10	0	0	0	24	0	0
	bush	0	0	0	0	3	0	0
	treeBig	1	0	0	0	0	0	0

Table 5.7: Confusion Matrix for Validation Data (locale veg2-1), Sensor = (snowball)

For the two non-zero classes, *grass* and *bramble*, we then compute evaluation metrics from the confusion matrix. The metrics we use are the recall and normalized precision. Recall is computed by dividing the main diagonal element by the sum of its column. Precision is computed by dividing the main diagonal element by the sum of its row. Normalized precision is a variant of precision that normalizes out the number of data points per class. First, the matrix is normalized by dividing each element by the sum of its column. From this normalized matrix, the precision is computed as before. Depending on the application, precision or normalized precision may be the appropriate choice. For instance, if the application relies on getting the most accuracy per volume of terrain, independent of the class, then precision would be more appropriate. In our application, the classes with fewer data points are often more important to classify because they are hazardous terrain. So we highlight them by this normalization process. Separate from whether they are hazardous or not, we do not want the limitations of our data collection procedure to affect the classification results. Table 5.8 presents the recall and normalized precision metrics for the confusion matrix in table 5.7. Note that for a binary classifier, 50% recall signifies no information. Across seven classes, this drops to 14.3%. So a 36% recall rate on grass, although not high, is still significant.

Class	Recall	Normalized Precision
grass	0.36	1.00
bramble	0.57	0.88

Table 5.8: Evaluation Metrics for Validation Data (locale veg2-1), Sensor = (snowball). Generated from confusion matrix 5.7.

The next step is to condense the metrics in table 5.8 down to one metric, which we call the *min of min*. We take the minimum recall of each class (in this case 0.36). And we take the minimum normalized precision for each class (in this case 0.88). And then we take the minimum of those two minimums (in this case 0.36). We repeat this process for all 380 iterations.

In summary, for each of the 380 sensor/data combinations, we:

1. train on training data (7 classes, SVM RBF binary nodes with whitening and hyperparameter tuning, combined with a 1v1 DDAG)
2. test on validation data
3. generate confusion matrix for validation data
4. turn confusion matrix into recall and normalized precision metrics
5. turn these metrics into one min of min metric

Table 5.9 shows the 10 out of the 380 iterations with the highest min of min values. The eleventh row is for the snowball sensor, described in the example above. The snowball sensor acts as a benchmark, since we used this in our early Gator work as well. Each of the top 10 rows is one of the 380 iterations. The first column shows the  $\binom{19}{1}$  or  $\binom{19}{2}$  sensor combination used for that iteration. (Refer to table 5.4 for the 19 sensor signals we are choosing from.) The second column denotes whether that iteration used a short

time window of 0.2 sec or 0.4 sec. The last column shows the min of min value. These are presented in descending order. The 0.36 value for the snowball sensor in the eleventh row is what we discussed above for computing the min of min from table 5.8.

Sensor Signal	Window length	Min of min
vt500-bumper, adxl-axle-up	0.20	0.88
vt500-axle, vt500-bumper	0.20	0.81
vt500-bumper, iepe10g-axle-up	0.20	0.80
vt500-bumper	0.40	0.79
vt500-bumper, imu-yAccel	0.20	0.73
contact-bumper-forward	0.40	0.69
contact-bumper-forward	0.20	0.66
adxl-axle-up, contact-bumper-forward	0.20	0.60
imu-yAccel, iepe10g-bumper-forward	0.20	0.60
vt500-bumper, imu-xAccel	0.20	0.58
snowball	0.20	0.36

Table 5.9: Top 10 Sensor Combinations and Accuracies. The eleventh row is for the snowball base sensor. “Min of min” stands for: minimum of [minimum recall across all classes, minimum normalized precision across all classes]. The value of the snowball sensor can be seen in table 5.8 as the minimum of all elements in the table. Similarly, the value of the optimal combination (vt500-bumper, adxl-axle-up) can be seen in table 5.11.

The rationale for using this min of min metric is that we want our classifier to be strong in both recall and normalized precision. By looking at the *min of min* instead of the *mean of mean*, we are trying to minimize the variance across classes.

The top choice (the top row of the table) is for the (*vt500-bumper, adxl-axle-up*) combination, with an 0.2 sec window. This is the vt500 waterproof air microphone mounted near the front bumper, along with the z-acceleration signal coming from the analog device unit mounted on the axle. This is the proprioceptive data that we will use moving forward for our full self-supervised framework. It makes intuitive sense that the combination would include an air microphone and a vibration sensor. It also makes intuitive sense that the combination would include one sensor from the bumper and once sensor from the axle along the vertical axis. It is interesting and useful to know that the expensive high-end IEPE sensors do not provide a benefit over the cheap ubiquitous ADXL accelerometers. However, this might be because the ADXL accelerometers were more sensitive. They were specified as having a dynamic range of  $\pm 3g$ , whereas even the more sensitive of the two IEPE varieties was still a high value of 10g. A future design iteration might involve looking for higher-end vibration sensors with a more sensitive range.

Tables 5.7 and 5.8 presented the confusion matrix and associated metrics for the benchmark snowball sensor. These are the numbers used for computing the 0.36 value in table 5.9. Tables 5.10 and 5.11 present the corresponding confusion matrix and metrics for the optimal (*vt500-bumper, adxl-axle-up*) combination. These are the numbers used for computing the 0.88 value in table 5.9. This is included here as a reference, in case the reader is inter-



ested in comparing the confusion between classes for the optimal sensor combination. In comparing the confusion matrix for the benchmark to the confusion matrix for the optimal combination, we can see a few trends. For instance, in the optimal combination, the bramble data is predicted at a much higher recall rate. In the benchmark, the bramble is often misclassified as vegSoft. It intuitively makes sense that the snowball air microphone would have trouble distinguishing these two above-ground terrain classes, whereas the vibration sensor from the optimal combination would be able to distinguish between the drastic differences in terrain compliance. The grass class also has a much higher recall rate for the optimal combination. In the benchmark, the grass is often misclassified as a number of the other classes. In the optimal combination, this happens much less often, resulting in a much higher recall rate. However, the grass is still misclassified as bramble at around the same rate, which explains why the normalized precision for bramble only improves by 4%.

		Actual Label						
		grass	grass-leaves	pavement	vegSoft	bramble	bush	treeBig
Predicted	grass	112	0	0	0	0	0	0
	grass-leaves	1	0	0	0	0	0	0
	pavement	1	0	0	0	0	0	0
	vegSoft	2	0	0	0	2	0	0
	bramble	11	0	0	0	40	0	0
	bush	0	0	0	0	0	0	0
	treeBig	0	0	0	0	0	0	0

Table 5.10: Confusion Matrix for Validation Data (locale veg2-1), Sensor Combination = (vt500-bumper, adxl-axle-up)

Class	Recall	Normalized Precision
grass	0.88	1.00
bramble	0.95	0.92

Table 5.11: Evaluation Metrics for Validation Data (locale veg2-1), Sensor Combination = (vt500-bumper, adxl-axle-up). Generated from confusion matrix 5.10.

## 5.6.2 Test Results

In section 5.6.1, we learned a multiclass model with the optimal sensor combination: (*vt500-bumper, adxl-axle-up*). We learned this by training on a set of locales that spanned the seven classes, and then validating on locale *veg2-1*. Using the same trained model, we now test on locale *bramble1-1*.

Table 5.12 shows the training and test locales used. This is similar to table 5.5, except the last row is the test locale instead of the validation locale. Table 3.6 from section 3.4 presented the number of proprioceptive human-labeled data points for each class in each locale. Working from tables 3.6 and 5.12, we present in table 5.13 the number of training

and test data points we have for each class. The column for the training data point numbers are the same as in table 5.6. The column for the test data point numbers are different, because they are for data in locale *bramble1-1* instead of locale *veg2-1*.

	Locale	Classes in that locale
<b>training</b>	sea2-11	grass, vegSoft
	wood1-7	bramble, grass
	bush4-1	bush, grass
	tree2-1	treeBig, grass-leaves, pavement
<b>test</b>	bramble1-1	bramble, pavement, grass-leaves

Table 5.12: Training and Test Locales for Proprioception Final Results

Class	Training	Test
grass	619	NA
grass-leaves	102	49
pavement	24	25
vegSoft	132	NA
bramble	94	41
bush	11	NA
treeBig	2	NA

Table 5.13: Number of Training and Test Data Points for Proprioception Final Results

Locale *bramble1-1* is the locale used for our final self-supervised results. In the self-supervised framework, this is the *current* locale, as presented in sections 4.2 and 4.3. This is the locale for which we will have proprioceptive predictions inside the robot’s path. These predictions will be used as labels for training the exteroceptive data.

Section 5.6.1 discussed how the model is trained, which we summarize here. We train a seven class multiclass classifier. The classes are: grass, grass-leaves, pavement, vegSoft, bramble, bush and treeBig. We train this model using the training data as presented above. We limit the number of data points by the smallest ground class, and for smaller classes, we repeat data points. The data points are short time windows of 0.2 second length, with 50% overlap between windows. For each short time window, we really have two data points, one from the *vt500-bumper* sensor and one from the *adxl-axe-up* sensor. We extract 9-d *gianna and shape* feature vectors from each window, and then concatenate them into an 18-d feature vector. These feature vectors are then fed into our model. The multiclass model is build off a *one-vs-one* DDAG, with an SVM for each binary node. Each binary node compares two out of the seven classes. Each binary node is an SVM with an RBF kernel, whose sigma and slack hyperparameters are automatically tuned through iterative cross-validation on each hyperparameter choice.

Once the model is trained, we make predictions on the test data. After predictions are made for each short time window, we smooth the predictions with a mode filter, as

described in section 5.5. We slide a larger window over the sequence of short time windows that is five times the size. We tally the votes for each data point's prediction in this larger window, and use this vote to relabel the data point in the center of the window.

Using these smoothed predictions on the test data from locale *bramble1-1*, we generate a confusion matrix, presented in table 5.14. The columns are the ground truth human labels for each class. The rows are the class predictions. Note that the three classes that exist in the test data are *bramble*, *pavement* and *grass-leaves*. These are the only classes with non-zero columns in the matrix. Also note that even though only these three classes show up at the test locale, the model is trained on all seven classes, and predictions can be made for any of these seven classes. For comparison, we also show the confusion matrix using the benchmark *snowball* sensor in table 5.15.

		Actual Label						
		grass	grass-leaves	pavement	vegSoft	bramble	bush	treeBig
Predicted	grass	0	0	0	0	0	0	0
	grass-leaves	0	35	0	0	0	0	0
	pavement	0	0	18	0	0	0	0
	vegSoft	0	0	0	0	0	0	0
	bramble	0	14	7	0	41	0	0
	bush	0	0	0	0	0	0	0
	treeBig	0	0	0	0	0	0	0

Table 5.14: Confusion Matrix for Test Data (locale *bramble1-1*), Sensor Combination = (vt500-bumper, adxl-axle-up)

		Actual Label						
		grass	grass-leaves	pavement	vegSoft	bramble	bush	treeBig
Predicted	grass	0	14	0	0	0	0	0
	grass-leaves	0	35	8	0	0	0	0
	pavement	0	0	13	0	0	0	0
	vegSoft	0	0	0	0	0	0	0
	bramble	0	0	4	0	41	0	0
	bush	0	0	0	0	0	0	0
	treeBig	0	0	0	0	0	0	0

Table 5.15: Confusion Matrix for Test Data (locale *bramble1-1*), Sensor = (snowball)

Using these two confusion matrices, we compute metrics to compare them. Table 5.16 shows the recall metrics. The first column denotes which class we are looking at. The second column are the recall rates for the benchmark *snowball* data points. The third column are the recall rates for the optimal (*vt500-bumper*, *adxl-axle-up*) data points. Each row is one of the three classes that shows up in the test data. The last row is the average recall across all classes for that sensor combination. Similarly, table 5.17 shows the normalized precision metrics for each data choice and test class, and then averaged across the classes.

Class	Recall (snowball)	Recall (vt500-bumper, adxl-axle-up)
grass-leaves	0.71	0.71
pavement	0.52	0.72
bramble	1.00	1.00
average	0.74	0.81

Table 5.16: Recall Comparison of Base Sensor (snowball) to Learned Sensor Combination (vt500-bumper, adxl-axle-up). Using Test Data (locale bramble1-1). Generated from confusion matrices 5.14 and 5.15.

Class	Normalized Precision (snowball)	Normalized Precision (vt500-bumper, adxl-axle-up)
grass-leaves	0.69	1.00
pavement	1.00	1.00
bramble	0.86	0.64
average	0.85	0.88

Table 5.17: Normalized Precision Comparison of Base Sensor (snowball) to Learned Sensor Combination (vt500-bumper, adxl-axle-up). Using Test Data (locale bramble1-1). Generated from confusion matrices 5.14 and 5.15.

We get a 7% improvement on recall, and a 3% improvement on normalized precision. The recall improves for each class separately. The normalized precision goes down for the bramble class. Looking back at the confusion matrices, this is happening mostly because the grass-leaves class is being falsely predicted as bramble for the optimal (*vt500-bumper, adxl-axle-up*) data choice. For the benchmark *snowball* data choice, the grass-leaves class gets falsely predicted as grass. Since grass is not one of the actual classes that show up in this locale, we cannot compute a precision or normalized precision value for it. This is one of the limitations of using precision or normalized precision as a metric for test data where not all of the classes are represented. As a result, the normalized precision for benchmark sensor is higher than it should be, relatively speaking with respect to the values for the optimal sensor combination. In turn, the 3% improvement in normalized precision is lower than it should be.

Nevertheless, we do get some improvement in both the recall and normalized precision metrics when transitioning from the benchmark sensor to the optimal sensor combination. This demonstrates that our learned sensor selection is able to generalize across locales. (We learned the sensor selection using the validation locale *veg2-1*, and we tested the sensor selection on the test locale, *bramble1-1*.) Moving forward in our self-supervised work, we will use this optimal sensor combination.

# Chapter 6

## Exteroception Module

In this chapter we present the exteroception module of the self-supervised framework. This is used for the second step of the two-step framework presented in chapter 4, figure 4.11. In that second step, predictions from the first step (the proprioception module) are used as training labels. In this chapter, we replace these proprioceptive predictions with human labels so that we can evaluate the performance of the exteroception without errors from the proprioception propagating through.

In section 6.1, we describe our implementation of an unsupervised feature learning technique using a Variational Auto Encoder (VAE) to learn the feature space. This is the feature vector we use on our exteroceptive data. Section 6.2 then describes the process of feeding these feature vectors into a multiclass model for training. In section 6.3 we present validation experiments and results that compare different feature and classifier parameter choices. This includes comparing the VAE feature vectors with other benchmark feature vectors. This also includes comparing the SVM kernel choice, the image patch size, and color versus grayscale image patches. Once the best feature choices and parameters have been chosen from this validation process, we present our test results for the exteroception along with the self-supervised results in chapter 7. These are presented as the vision floor and the vision ceiling benchmarks, to be compared against the self-supervision.

### 6.1 Unsupervised Feature Learning

For our exteroceptive feature extraction, we learn our feature vectors instead of surveying various feature extraction techniques as we did for the proprioception. Recent deep learning methods allow us to do this with the Variational Auto Encoder (VAE) [20], [32]. This is considered a state-of-the art method for computer vision applications in deep learning [36]. The VAE is an unsupervised feature learning technique which optimizes the full representation of the data in the feature space. We use the implementation from the Tensor Flow library that came out of the Google Brain project [1].

Figure 6.1 depicts a VAE from a high level. It consists of two sequential convolutional neural networks. The first in the sequence is an inference network, which takes image patches as input data points, and outputs them into a latent feature space. The second

network is the generative network, which takes these latent feature vectors as input, and then for each data point, it outputs a reconstructed version of the original image patch. The closer the reconstruction, the better the model is working. A loss function is used in the learning process to compare each input image patch to its corresponding output. The lower the loss, the closer the reconstruction. If the intermediate latent variables are able to *encode* the information from the original image, then a generative network can be learned that reconstructs the images with high quality. A set of unlabeled data points is fed into the VAE, and during each iteration of the optimization process, the loss function is computed between the input and output images. The training is finished once this loss has been minimized.

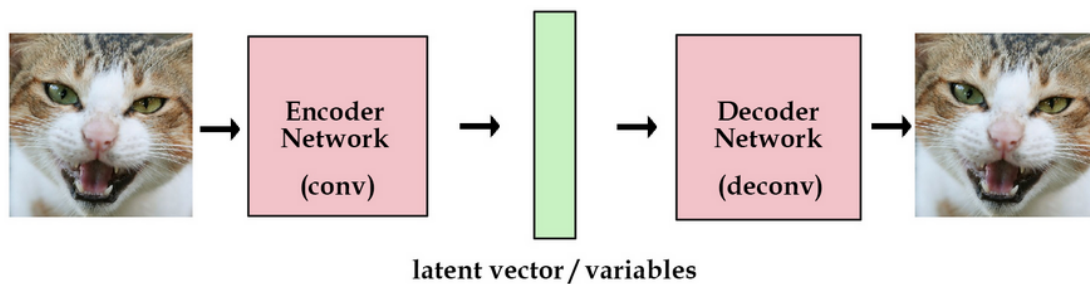


Figure 6.1: Variational Auto Encoder From a High Level. Taken from <http://kvfrans.com/variational-autoencoders-explained/>

After the VAE has been trained, we can then use the intermediate latent vectors as our feature vectors. Considering the pipeline in figure 6.1, it is just the left part of the pipeline that will be used after training is complete. The left image patch of the cat is one input data point, and the middle green rectangle is a feature vector for that data point. By passing each input image patch through the trained encoder network (the left pink square), we generate a feature vector for that image patch.

This unsupervised step is just for learning the feature space. Once the feature space has been learned, the feature vectors are used in a supervised model within our overall framework. The feature vectors will be tagged with self-supervised labels from the proprioceptive predictions, and these are fed into a supervised multiclass classification algorithm, as discussed next in section 6.2.

The unsupervised data that we used for training our VAE came from the locales listed in table 6.1. This data included terrain from all seven classes in the multiclass model that we use for our self-supervised results: grass, grass-leaves, pavement, soft vegetation, bramble, bushes and trees. However, the VAE is unsupervised, so it is unaware of these class labels.

Locale	Classes in that locale
sea2-11	grass, vegSoft
wood1-7	bramble, grass
wood1-8	bramble, grass
bush4-1	bush, grass
tree2-1	treeBig, grass-leaves, pavement
veg2-1	bramble, grass

Table 6.1: Locales and Terrain Classes in Each Locale Used for VAE

The data came from Pixim images at various stopped intervals along the robot’s path. Figure 6.2 depicts the robot at the start of a trial. (This is the same as figure 3.8 presented earlier.) The robot makes a straight path as it drives over ground terrain, and then it drives into above-ground at the end of the path. Along the path, it stops at various waypoints, as shown in the diagram. The Pixim images used are from these stopped waypoints. Although we have data from both the left and right cameras, we experimented only with images from the left camera, without loss of generality. So, for instance, at a particular locale, we might consider 5 waypoints. The waypoints might be 8m, 6m, 4m, 2m and 0m away from the above-ground terrain. At each waypoint, we use one Pixim image from the left camera. Refer to appendix section 9.2.1 to see images from the left camera at various waypoints in each locale. (Note that the cameras are slightly cross-eyed, so the left camera is pointed slightly to the right. Hence, the above-ground terrain at the center of the robot’s field of view is on the left side of the image.) So we have a total of 5 images from each locale. We are considering 6 locales, as listed in table 6.1, so then we have 30 images in our dataset. Each data point is a local image patch centered around each pixel. We consider any pixel in an image as the center of a data point. Each image has a resolution of 487 by 720 pixels. So there are  $487 * 720 = 350640$  pixels, or data points, in an image. Across 30 images, this is  $350640 * 30 = 10519200$  data points. This is around 10.5 million data points. To conserve computational time, we randomly sample 60,000 data points out of these 10.5 million to use as training data for the VAE.

We set the dimension of the latent feature vector to 50. The higher the dimension, the more room there is for the vector to *encode* the data space. But then the higher the dimension of our feature vectors for the next step of our algorithm. We set the number of epochs of the learner to 100. This means the algorithm will perform 100 optimization steps to try to minimize the loss function. The TensorFlow tutorial application that builds a VAE uses the MNIST data set, with grayscale binarized image patches of numbers 0 through 9. We modify the code to allow for 3-channel RGB non-binarized image patches to be the data input. The inference network has two convolutional layers. The first layer has an output of 32 filters and the second layer an output of 64 filters. Each convolutional layer runs a convolution kernel window over the input. We experiment with different patch sizes for our input data patches, and different kernel sizes. For a 28 x 28 patch size, we use a 3 x 3 kernel. For a 54 x 54 patch size, we use a 6 x 6 kernel. The generative network mirrors the two convolutional layers with two corresponding deconvolutional layers.

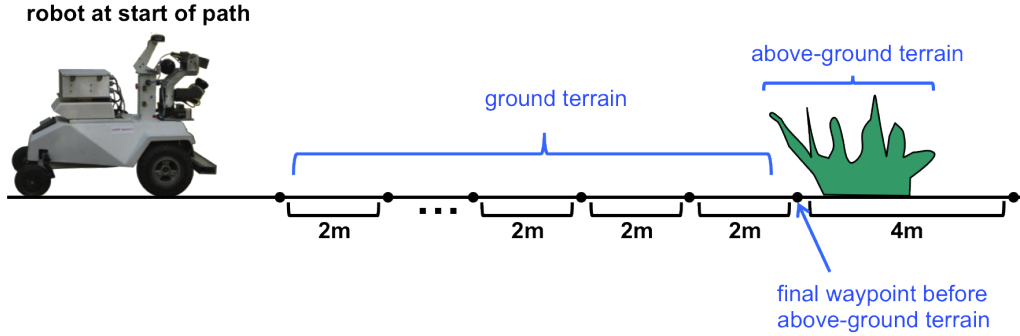


Figure 6.2: Robot Viewing Scene from Stopped Intervals Along Path

One of the benefits of a VAE is that the user can look at the reconstructed images and get a qualitative sense of how well the model has been trained. If the reconstructed image patches retain the distinguishing qualities of their corresponding input image patches, then the VAE is working properly. Figure 6.3 shows an example of 10 data points for the bramble class. The top row are the input image patches. The bottom row are the decoded output image patches. Each output patch is directly below its corresponding input. Refer to appendix section 9.1 to see 100 data point examples for each of the seven terrain classes, and their corresponding decoded outputs. Both in this figure and in the appendix, we used a  $28 \times 28$  patch size with a  $3 \times 3$  kernel. Note that we are using the terrain class label to qualitatively group the data points here for our qualitative evaluation. These labels are not used in the training of the VAE.

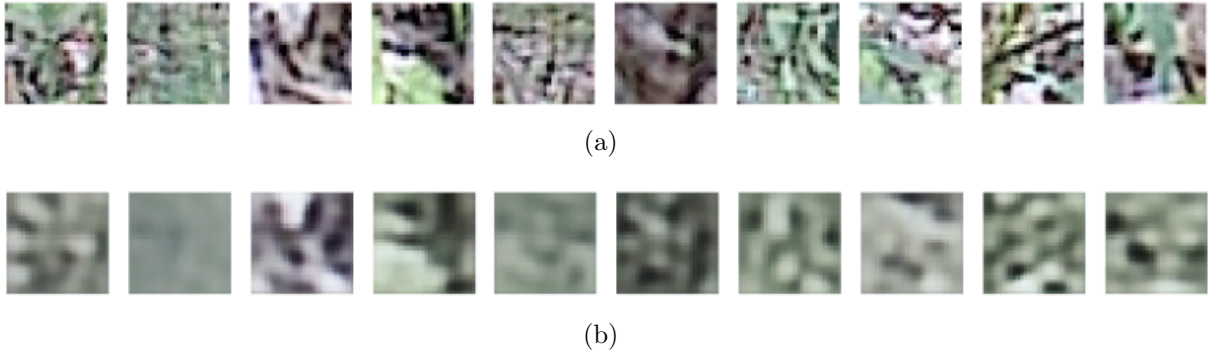


Figure 6.3: Example of 10 Data Points for the Bramble Class. *Top*: input images to the VAE. *Bottom*: Corresponding decoded outputs of the VAE. Each output patch is directly below its corresponding input image patch.

## 6.2 Classification

Our multiclass classifier is similar to what we use for our proprioception module. The multiclass framework uses a standard *one-vs-one* approach: we train binary Support Vector Machines (SVM's) between each pair of classes, and then each binary classifier is a node



in a Decision Directed Acyclic Graph (DDAG), as described in Platt et. al. [29]. Each binary node is an SVM, implemented in the SVM-Light library (Joachims [18]). The data points are local image patches, centered around a particular pixel. The center pixels used are the subset of pixels from the images to which Wobbler points are mapped. The input feature vectors to the SVM are the latent feature vectors from the Variational Auto Encoder (VAE), described in section 6.1. We also experiment with Gabor ([13]) and RGB feature vectors as benchmarks to the VAE. These different benchmark feature choices are discussed inline with our experiments in section 6.3.

The multiclass classifier works on some subset of the seven terrain classes: grass, grass-leaves, pavement, soft vegetation, bramble, bushes and trees. There are three experiments where we run the exteroception module: the second step of the self-supervision, the benchmark vision floor, and the benchmark vision ceiling. These are presented in section 4.3, in tables 4.2, 4.3 and 4.4, respectively. For the floor experiments, the class set is all seven classes. For the ceiling experiments, the class set is the subset of classes that occur inside the robot’s path at the current locale. For the second step of self-supervision, the class set is the subset of classes that is *predicted* by the proprioception to occur inside the robot’s path at the current locale. Whatever subset of classes is used, the multiclass classifier is still built with the DDAG *one-vs-one* graph.

For each binary node, we zero-mean shift and whiten the data before passing it into the model to be trained. These transformations are stored as part of the model for that node, and then used as well on the test data. For each binary node, we experiment with both linear and radial basis function (RBF) kernels. RBF kernels sometimes perform better than linear kernels because they can handle non-linear separations. Linear kernels however are sometimes better for high dimensional feature spaces. The hyperparameters for the kernels must be tuned. This includes the slack hyperparameter for both kernels and the sigma hyperparameter for the RBF kernel. Descriptions of what these hyperparameters mean are discussed in section 5.4. As we do with the proprioception module, we tune the hyperparameters through an automated cross-validation process. For each binary node, we loop through different combinations of hyperparameter values. For each combination, we train an SVM on half of the training data, and then compute recall rates for the other half of the data. We then average the recall rate of the two classes in that binary node. The combination of hyperparameters with the highest accuracy is then used for our final model.

Deep learning is considered state of the art for machine learning applications in computer vision. Our learning process is really a combination of the VAE, which is a deep learning technique, followed by the DDAG SVM described here. So one can think of the DDAG SVM as a final layer to the convolutional neural network in the VAE. An alternative approach would be to use a supervised Convolutional Neural Network (CNN). Most supervised deep learning methods benefit from large data sets. We are working with small datasets which we collect and hand-label ourselves. Our exteroceptive data sets are larger than our proprioceptive data sets, but they are still small in relation to common machine learning data sets. SVM’s are a good choice for handling small datasets because they are less prone to overfitting than other methods. By using an unsupervised feature learning approach with the VAE, we are able to leverage the advantages of deep learning without

the constraints of a minimal amount of hand labels. We then top it off with a supervised SVM layer.

### 6.3 Experiments and Results

In this section, we run experiments on different feature vectors and classifiers for the exteroception module. These are supervised experiments, using human labels for training. Human labels are also used for evaluating the performance of the predictions. We use data from the locales listed in table 6.2. This is a subset of the locales used as training data in our final experiments in chapter 7. This data is split up into training and validation data for these experiments.

Locale	Classes in that locale
sea2-11	grass, vegSoft
wood1-7	bramble, grass
tree2-1	treeBig, vegDry, grass-leaves, pavement

Table 6.2: Locales and Terrain Classes in Each Locale Used for Exteroceptive Feature Comparison

For each of these locales, we look at various stopped intervals that the robot makes along its path. At each stopped interval, the Wobbler makes a full 3-second revolution, collecting a full point cloud from that vantage point. We consider the subset of Wobbler points that fall within a range of [0.8m, 50.0m]. 50.0m is the maximum range limit of the Sick LMS 151 laser specifications. The minimum range is 0.5m, but we found through experiments that 0.8m is a safer minimum range to use, in order to avoid faulty data. This subset is then projected onto the camera image plane. The pixels onto which they are projected are the center of each data point. The data points are local image patches centered around these pixels.

We train on human-labeled image patches inside the robot’s path, and validate on image patches outside the robot’s path, using human labels on the validation data to evaluate the performance of the predictions. Figure 6.4 shows the projected Wobbler points inside and outside of the robot’s path. This is at some stopped interval along the path, looking at the left and right pair of camera images from that stopped interval, and the corresponding point cloud from that same stopped interval projected onto the image plane. The points inside the robot’s path on the left are the center pixels of the training data, and the points outside the robot’s path on the right are the center pixels of the validation data. Section 4.1 describes the process of determining the subset inside the path and the subset outside the path. This involves using the robot’s pose estimates to create rectangular footprint prisms. These are used in the self-supervised framework for registering the proprioceptive data, but here we ignore the proprioceptive data. We simply use the rectangular footprints with human labels to create the subsets for our training and test data (inside and outside the robot’s path).

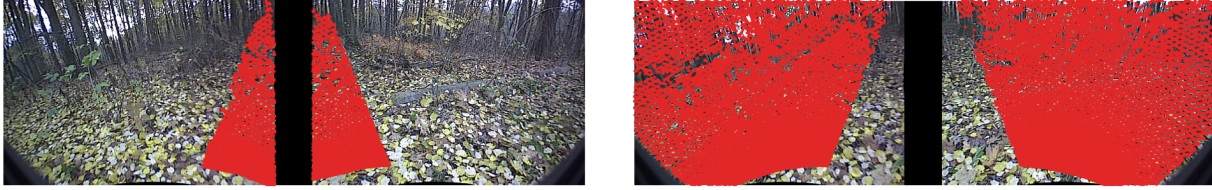


Figure 6.4: Exteroceptive Data Inside and Outside the Robot’s Path. *Left*: Inside the robot’s path. *Right*: Outside the robot’s path. Each data point is a local image patch centered around a pixel. The center pixels are shown as red dots. They are the pixels to which Wobbler points are mapped.

Table 6.3 lists the number of training and validation data points for each terrain class at each locale. Remember that for each locale, we have multiple stopped intervals that the robot makes along its path (refer to figure 6.2). The image pair above in figure 6.4 shows one such stopped interval. The full set of training and validation data for that locale comes from the images at all of the stopped intervals, inside and outside the robot’s path, respectively. Note that we only use images from the left camera in our experiments.

Locale	Class	Training/Inside Path	Validation/Outside Path
sea2-11			
	grass	6052	15353
	vegSoft	4623	10677
wood1-7			
	grass	4579	15171
	bramble	958	8074
tree2-1			
	grass-leaves	4900	15725
	pavement	5079	15811
	treeBig	1450	1031

Table 6.3: Number of Training and Validation Data Points for Exteroceptive Ceiling Experiments

We ran experiments with different feature and classifier combinations. For each experiment, we looked at one locale, and chose two of the classes from that locale to create a binary classifier. So if a locale had more than two classes, then we ran separate experiments for each of the pairs of classes. For each experiment, we looked at one type of feature vector with one type of classifier.

These experiments are similar to the vision ceiling experiments outlined in table 4.4. The only difference is that instead of training a multiclass classifier for the classes in that locale, we consider each pair of classes. This is analogous to separately examining the binary nodes from the multiclass *one-vs-one* Decision Directed Acyclic Graph (DDAG). We can think of this, in a sense, as a ceiling of the ceiling, because the one-vs-one scheme will only decrease the performance of the recalls and precisions for each class. Our goal here

is to raise the ceiling as much as possible, by validating the performance of different feature and classifier combinations, and choosing the best combination. We discuss these feature and classifier combinations below. The actual ceiling experiments used for benchmarking our final self-supervised results are presented in chapter 7.

Note that the data we use here to evaluate the classifier predictions is referred to as validation data rather than test data. We reserve the term *test data* for predictions in our final results. The data used here is to validate different feature and classifier choices, so this is a tuning stage of our overall model. We *select* feature and classifier parameters here, similarly to how we used validation data to learn, or select, sensor combinations for our proprioception module in section 5.6.1. The validation data used here is from different locales than the test data that we use in our final results.

For the classifiers, we experiment with Support Vector Machines (SVM’s) with linear kernels and Radial Basis Function (RBF) kernels. For each trained model, we use the same number of data points from both classes. We use the number of data points from the smaller of the two classes, and then for the larger class, we randomly sample from the set of points to get this smaller subset. As described earlier, the features passed in for training are zero-mean shifted and whitened, and these parameters are stored as part of the model, so that the validation data can be transformed in the same fashion. Also as described earlier, we tune the sigma and slack hyperparameters with an automated cross-validation process.

For feature vectors, we experimented with the features extracted from the Variational Auto Encoder (VAE) as well as Gabor ([13]) and RGB features. Both the Gabor and RGB features are used as benchmark comparisons to the state of the art VAE approach. We chose to experiment with Gabor features because they are a traditional feature choice used for texture discrimination. By RGB feature, we simply mean the average of each R, G, and B channel across the image patch. We chose to experiment with RGB features because they are the simplest, lowest-dimensional color information that we can acquire. Gabor features are traditionally grayscale, but they can be created in color as well. We experimented with both. We also experimented with concatenating a grayscale Gabor feature vector together with an RGB vector.

For the VAE features, we experimented with 28x28 patch sizes, as well as 54x54 patch sizes. The 28x28 patch size is what is used in the tensor flow tutorial example, so we started with this. We then experimented with a 54x54 patch size to see if a larger patch would give us more information. The 28x28 patches have 3x3 convolution kernel windows. The 54x54 patches have 6x6 convolution kernel windows. We experiment with both color and grayscale versions of the VAE.

The VAE vector is 50-d. The Gabor vector is 144-d. This is dependent on the various kernels and rotations that are used for creating it. The RGB vector is 3-d (one for each R, G, B channel). When we concatenate a Gabor and RGB vector, it is then 147-d.

The full list of feature/classifier combinations is given in table 6.4, along with the explanations for how we are abbreviating their names.

<b>Feature/Classifier</b>	<b>Description of Abbreviation</b>
VAE 28 Color RBF	convolutional variational auto encoder feature vector, 28 x 28 image patch, in color, support vector machine with a radial basis function kernel
VAE 28 Color Linear	convolutional variational auto encoder feature vector, 28 x 28 image patch, in color, support vector machine with a linear kernel
VAE 54 Color RBF	convolutional variational auto encoder feature vector, 54 x 54 image patch, in color, support vector machine with a radial basis function kernel
VAE 54 Color Linear	convolutional variational auto encoder feature vector, 54 x 54 image patch, in color, support vector machine with a linear kernel
VAE 28 Grayscale RBF	convolutional variational auto encoder feature vector, 28 x 28 image patch, in grayscale, support vector machine with a radial basis function kernel
VAE 28 Grayscale Linear	convolutional variational auto encoder feature vector, 28 x 28 image patch, in grayscale, support vector machine with a linear kernel
Gabor 28 Grayscale RBF	Gabor feature vector, 28 x 28 image patch, in grayscale, support vector machine with a radial basis function kernel
Gabor 28 Grayscale Linear	Gabor feature vector, 28 x 28 image patch, in grayscale, support vector machine with a linear kernel
Gabor 28 Grayscale + RGB 28 RBF	Gabor feature vector in grayscale, concatenated with an RGB vector, 28 x 28 image patch, support vector machine with a radial basis function kernel
Gabor 28 Grayscale + RGB 28 Linear	Gabor feature vector in grayscale, concatenated with an RGB vector, 28 x 28 image patch, support vector machine with a linear kernel
Gabor 28 Color RBF	Gabor feature vector, in color, 28 x 28 image patch, support vector machine with a radial basis function kernel
Gabor 28 Color Linear	Gabor feature vector, in color, 28 x 28 image patch, support vector machine with a linear kernel
RGB 28 RBF	RGB vector, 28 x 28 image patch, support vector machine with a radial basis function kernel
RGB 28 Linear	RGB vector, 28 x 28 image patch, support vector machine with a linear kernel

Table 6.4: List of Feature/Classifier Combinations on which We Run Experiments

For each trial, we train on the data inside the robot’s path and validate on the data outside the robot’s path. Human labels are used on the validation data to evaluate performance. A 2x2 confusion matrix will result from the binary classifier validation predictions for each trial. We compute recall and precision metrics for each class. The recall values are presented in table 6.5, and the precision values are presented in table 6.6. The class names are abbreviated, so that each table can fit on one page. *bram* stands for bramble, *pave* stands for pavement, and *gr-le* stands for grass-leaves. Each pair of rows in the table denotes an experiment at that locale. The left-most column is the locale we’re looking at, and the next column are the classes in that locale. So for example, the first two rows are a binary classifier at locale *wood1-7*, for classes grass and bramble. The first row shows the recall (or precision) values for grass. The second row shows the recall (or precision) values for bramble. Starting from the third column, each column represents an experiment for a particular feature/classifier combination. The abbreviations for the column names were presented above in table 6.4. The number in each cell is the recall (or precision) value for a particular class, at a particular locale, for a particular feature and classifier combination. The bottom row is the average of that column. This is the average across all locales and classes for a particular feature/classifier combination. The order of the feature columns is pretty much in descending order of this average value, for both recall and precision. We can see that the color VAE performs better than the grayscale VAE, or any of the Gabor choices. The VAE 28 Color RBF is what we’ll use moving forward for our self-supervised scheme. The VAE 28 Color performs the same whether we use the RBF or linear kernels, but we choose the RBF kernel so that we have the flexibility to make nonlinear cuts with our SVM hyperplane. This could potentially help with unforeseen nonlinearities in the test data.

<i>Trial</i>	<i>Class</i>	<i>VAE 28 Color RBF</i>	<i>VAE 28 Color Linear</i>	<i>VAE 54 Color RBF</i>	<i>VAE 54 Color Linear</i>	<i>VAE 28 Grayscale RBF</i>	<i>VAE 28 Grayscale Linear</i>	<i>Gabor 28 Grayscale RBF</i>	<i>Gabor 28 Grayscale Linear</i>	<i>Gabor 28 Grayscale + RGB 28 RBF</i>	<i>Gabor 28 Grayscale + RGB 28 Linear</i>	<i>RGB 28 RBF</i>	<i>RGB 28 Linear</i>		
wood1-7	grass	0.83	0.89	0.93	0.95	0.81	0.83	0.36	0.75	0.45	0.85	0.78	0.75	0.88	0.90
	bram	0.79	0.74	0.72	0.72	0.67	0.67	0.87	0.58	0.93	0.73	0.56	0.70	0.52	0.81
tree2-1	pave	0.98	0.97	0.95	0.75	0.97	0.96	0.64	0.69	0.65	0.68	0.74	0.69	0.39	0.65
	gr-le	0.92	0.88	0.91	0.88	0.89	0.80	0.96	0.42	0.98	0.75	0.97	0.77	0.90	0.88
tree2-1	pave	0.96	0.96	0.93	0.97	0.96	0.96	0.73	0.95	0.76	0.97	0.80	0.97	0.96	0.96
	treeBig	1.00	1.00	1.00	1.00	1.00	0.98	0.99	0.93	1.00	0.99	0.98	0.90	0.73	0.99
tree2-1	vegDry	0.88	0.90	0.98	0.99	0.88	0.89	0.88	0.80	0.97	0.98	0.96	0.95	0.99	0.99
	treeBig	0.95	0.97	0.39	0.62	0.91	0.94	0.84	0.91	0.72	0.75	0.77	0.44	0.41	0.72
sea2-11	grass	0.86	0.87	0.83	0.73	0.55	0.30	0.81	0.42	0.81	0.70	0.47	0.69	0.64	0.72
	vegSoft	0.72	0.78	0.81	0.63	0.49	0.54	0.17	0.54	0.19	0.62	0.90	0.61	0.77	0.63
tree2-1	vegDry	0.86	0.85	0.92	0.90	0.83	0.62	0.92	0.38	0.95	0.82	0.93	0.87	0.91	0.91
	pave	0.96	0.92	0.99	0.90	0.95	0.96	0.69	0.73	0.72	0.76	0.77	0.74	0.52	0.75
tree2-1	gr-le	0.97	0.97	0.98	1.00	0.95	0.94	0.90	0.84	0.99	0.98	0.98	0.93	0.99	0.99
	treeBig	0.95	0.97	0.58	0.85	0.88	0.93	0.79	0.90	0.73	0.84	0.79	0.61	0.24	0.85
<b>average</b>		<b>0.90</b>	<b>0.90</b>	<b>0.85</b>	<b>0.85</b>	<b>0.84</b>	<b>0.81</b>	<b>0.75</b>	<b>0.70</b>	<b>0.77</b>	<b>0.82</b>	<b>0.81</b>	<b>0.76</b>	<b>0.70</b>	<b>0.84</b>

Table 6.5: Recall Comparison, Validation Data. Naming convention of column headers: feature, patch size, color vs grayscale, Support Vector Machine kernel. For example, VAE 28 Color RBF = Variational Auto Encoder, 28 x 28 patch size, in color, Radial Basis Function kernel

Trial	Class	VAE 28 Color RBF	VAE 28 Color Linear	VAE 54 Color RBF	VAE 54 Color Linear	VAE 28 Grayscale RBF	VAE 28 Grayscale Linear	Gabor 28 Grayscale RBF	Gabor 28 Grayscale Linear	Gabor 28 Grayscale + RGB 28 RBF	Gabor 28 Grayscale + RGB 28 Linear	RGB 28 RBF	RGB 28 Linear		
wood1-7	grass	0.88	0.87	0.86	0.86	0.82	0.82	0.84	0.77	0.92	0.85	0.77	0.82	0.77	0.90
	bram	0.71	0.79	0.84	0.88	0.65	0.67	0.42	0.55	0.47	0.72	0.58	0.60	0.70	0.82
tree2-1	pave	0.92	0.89	0.92	0.87	0.90	0.83	0.94	0.54	0.97	0.73	0.97	0.75	0.80	0.85
	gr-le	0.98	0.97	0.94	0.78	0.96	0.95	0.73	0.57	0.74	0.70	0.79	0.71	0.59	0.72
tree2-1	pave	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.98	1.00
	treeBig	0.63	0.62	0.48	0.68	0.62	0.64	0.20	0.54	0.21	0.67	0.24	0.66	0.55	0.61
tree2-1	vegDry	0.99	0.99	0.92	0.95	0.99	0.99	0.98	0.99	0.96	0.97	0.97	0.92	0.92	0.96
	treeBig	0.53	0.57	0.73	0.94	0.52	0.54	0.49	0.39	0.78	0.83	0.73	0.53	0.83	0.89
sea2-11	grass	0.82	0.85	0.86	0.74	0.61	0.49	0.58	0.57	0.59	0.73	0.87	0.71	0.80	0.74
	vegSoft	0.79	0.80	0.77	0.61	0.43	0.35	0.37	0.39	0.41	0.59	0.54	0.57	0.60	0.61
tree2-1	vegDry	0.92	0.84	0.98	0.81	0.89	0.88	0.59	0.40	0.62	0.62	0.65	0.61	0.47	0.63
	pave	0.94	0.93	0.96	0.95	0.92	0.84	0.95	0.71	0.97	0.90	0.96	0.92	0.93	0.95
tree2-1	gr-le	1.00	1.00	0.97	0.99	0.99	1.00	0.99	0.99	0.98	0.99	0.99	0.97	0.95	0.99
	treeBig	0.66	0.69	0.60	0.96	0.52	0.52	0.35	0.27	0.81	0.79	0.72	0.37	0.71	0.80
average		0.84	0.84	0.85	0.86	0.77	0.75	0.67	0.62	0.74	0.79	0.77	0.73	0.76	0.82

Table 6.6: Precision Comparison, Validation Data. Naming convention of column headers: feature, patch size, color vs grayscale, Support Vector Machine kernel. For example, VAE 28 Color RBF = Variational Auto Encoder, 28 x 28 patch size, in color, Radial Basis Function kernel

Figure 6.5 shows examples of classifier predictions mapped onto the camera images. This is for a trial at locale *tree2-1*, running a binary classifier on the pavement and tree classes. The accuracies for this trial are given in rows five and six of tables 6.5 and 6.6. The first row of images shows some training data. The left image is pavement data. The right image is tree data. These are colored to show the human labels on the training data. These are Wobbler points inside the robot’s path. These are images from the left camera, which is pointed slightly to the right. So the robot’s path, which is in the center of the robot’s field of view, shows up along the left side of the images. The second row shows the predictions on some validation data from the *Gabor 28 Color RBF* feature/classifier combination. The left and right images are the same images as shown in the row above them for the training data, but now we are looking at the points outside the robot’s path. The colors show the class predictions. The third row shows the predictions on the same validation data, but now with the *VAE 28 Color RBF* combination. Remember that there



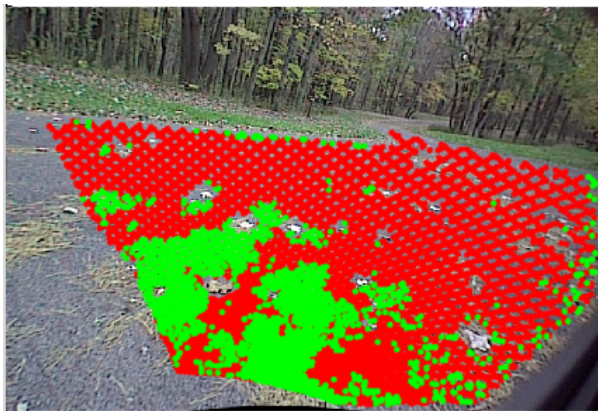
are multiple images used for training and testing, one image from each stopped interval waypoint along the robot's path. Only two images are shown here. The left column shows a stopped interval that is 2 meters away from the center tree inside of the robot's path. This is towards the end of the robot's path, when it's close to hitting the tree. This image is used for labeling and evaluating tree data. The right column shows a stopped interval that is 12 meters away from the center tree. This is further back in time, towards the beginning of the path. This image is used for labeling and evaluating pavement data, since there is a patch of pavement terrain in front of it that it will subsequently drive over.



(a) pavement training data



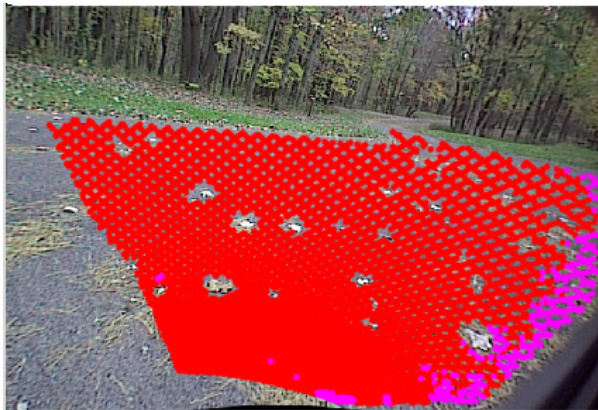
(b) treeBig training data



(c) pavement predictions, Gabor 28 Color RBF (red = true positives, green = false negatives)



(d) treeBig predictions, Gabor 28 Color RBF (cyan = true positives, magenta = false negatives)



(e) pavement predictions, VAE 28 Color RBF (red = true positives, magenta = false negatives)



(f) treeBig predictions, VAE 28 Color RBF (cyan = true positives, no false negatives)

Figure 6.5: Example Training and Validation Data from the *tree2-1* locale, with the (pavement, treeBig) Binary Classifier. Top row shows training data. Second row shows predictions using Gabor features. Third row shows predictions using VAE features. Both the second and third rows extract features from 28 x 28 RGB image patches. Both feature types are fed into an SVM with an RBF kernel.

We can observe that the *Gabor 28 Color RBF* combination in the second row performs poorly, while the *VAE 28 Color RBF* combination in the third row performs well. One can see from the images on the left that there is hay scattered across the pavement. We treat this as a visual variation in the pavement. In other words, we want the ground to still be classified as pavement even if it is covered with hay. In the top left image, we are showing that we pass in training data points for bare pavement as well as pavement that has hay on top of it. All of this is being labeled in the training step as pavement. So once the classifier is trained, then both types of data points in the validation set (data points for bare pavement as well as data points with pavement covered by hay) should be predicted as pavement. In the second row, on the left image, there are a lot of green false negatives, and these false negatives correspond to the patches of pavement that are covered by hay. This shows that the classifier is not able to handle both pavement variations. Since the SVM has an RBF kernel, it should be able to handle nonlinearity in the feature space, so most likely the problem is in the feature space itself. The top left image shows that there is more training data for the bare pavement than there is for the pavement covered by hay. So this discrepancy in the data sampling of these different terrain classifications, together with the limitations in the feature vector's ability to represent these variations, is too much for the classifier to handle. On the other hand, the VAE features in the third row do an excellent job at handling this variation. The only false negatives on the left bottom image are towards the very edge of the right side of the image (shown in magenta). These errors are most likely due to boundary issues, or perhaps camera blur towards the outer edges of the field of view. These types of errors could be eliminated by restricting the subset of Wobblers points considered to be within a slightly narrower field of view. These images present a nice example of how the VAE features are able to handle variations within the terrain of a particular class. As mentioned above, this is the feature/classifier combination that we use for our self-supervised test experiments in the next chapter. These are the VAE feature vectors, with a 28 x 28 patch size, in color, with an RBF kernel on the binary SVM nodes.



# Chapter 7

## Final Experiments and Results

We introduced the four main experiments that we would run in section 4.3. These included experiments for: the proprioception, the self-supervision, the vision floor and the vision ceiling. Here we present the results from running these experiments. We summarized the setup for each of the four experiments in tables 4.1, 4.2, 4.3, and 4.4. We repeat these tables inline here as we present the results of each experiment.

Table 7.1 lists the locales and the classes within each locale that are used for these four experiments. Each locale has a subset of the seven classes. The set of previous locales spans all seven classes. The current locale, *bramble1-1*, is where the self-supervision is tested. The breakdown for what data is used for training and testing is delineated below for each experiment. Note that the terrain class *treeBig* is listed in parentheses for the current locale. This is because it only shows up in the data outside the robot’s path.

	<b>Locale</b>	<b>Classes in that locale</b>
<b>previous locales</b>	sea2-11	grass, vegSoft
	wood1-7	bramble, grass
	bush4-1	bush, grass
	tree2-1	treeBig, grass-leaves, pavement
<b>current locale</b>	bramble1-1	bramble, pavement, grass-leaves, (treeBig)

Table 7.1: Training and Test Locales. Refer to appendix images 9.15, 9.16, 9.17, 9.18, 9.19, 9.20, 9.21, 9.22, 9.23, 9.24, 9.25 for visual data.

### 7.1 Proprioception

The results for the proprioceptive experiment are presented in section 5.6.2, with respect to evaluating the performance of the optimal learned sensor combination. We summarize these results here, with respect to the other three main experiments. Table 7.2 summarizes the setup for this experiment. (This is repeated from table 4.1.) The training data comes from the set of previous locales in table 7.1. The test data comes from the current locale, *bramble1-1*. The test predictions from the proprioception module at the current locale are then used to perform self-supervision at the current locale. We train a multiclass model

with 7 terrain classes: 3 ground classes and 4 above-ground classes. These include grass, grass-leaves, pavement, soft vegetation, bramble, bushes, and trees. We test on the classes that show up in the current locale, inside the robot’s path: grass-leaves, pavement and bramble.

<ul style="list-style-type: none"> <li>• data source: time series signals from microphones and vibration sensors</li> <li>• data point format: consecutive overlapping short time windows from the time series signals</li> <li>• multiclass model: classes A, B, C, D (in reality, 7 classes)</li> <li>• training data <ul style="list-style-type: none"> <li>▪ labels for training: human (marked sequences in time series signal)</li> <li>▪ location: previous locale(s), inside path</li> </ul> </li> <li>• test data <ul style="list-style-type: none"> <li>▪ labels for evaluation: human (marked sequences in time series signal)</li> <li>▪ location: current locale, inside path</li> </ul> </li> </ul>	
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--

Table 7.2: Proprioception Overview

The number of training and test data points is given in table 7.3. (This is repeated from table 5.13.) The labels come from human-labeled time sequences in the proprioceptive signals. We limit the training data set for each class to the number of data points in the smallest *ground* class. We repeat data points for smaller classes. Refer to sections 5.2, 5.3 and 5.4 for a discussion of the pipeline of turning proprioceptive signals into data points, then into feature vectors, and then training them with a multiclass classifier.

Class	Training	Test
grass	619	NA
grass-leaves	102	49
pavement	24	25
vegSoft	132	NA
bramble	94	41
bush	11	NA
treeBig	2	NA

Table 7.3: Number of Training and Test Data Points for Proprioception Final Results

Table 7.4 is the confusion matrix for the test predictions of the proprioception. (This is repeated from table 5.14.) This is for the optimal combination of proprioceptive sensor

signals from the set of 19 signals, (vt500-bumper, adxl-axle-up). This is a combination of an electret air microphone mounted a little above the bumper, and the z-axis acceleration coming from a cheap but sensitive accelerometer mounted near the wheel axle. Refer to section 5.6.1 for a discussion of how we chose this optimal combination. The columns in the matrix are for the ground truth human labels for the test data points. The rows are for the predicted labels. There are only a subset of terrain classes that show up at the current locale: grass-leaves, pavement, and bramble. The other classes that do not show up have columns of zero's in the matrix.

		Actual Label						
		grass	grass-leaves	pavement	vegSoft	bramble	bush	treeBig
Predicted	grass	0	0	0	0	0	0	0
	grass-leaves	0	35	0	0	0	0	0
	pavement	0	0	18	0	0	0	0
	vegSoft	0	0	0	0	0	0	0
	bramble	0	14	7	0	41	0	0
	bush	0	0	0	0	0	0	0
	treeBig	0	0	0	0	0	0	0

Table 7.4: Confusion Matrix for Test Data (locale bramble1-1), Sensor Combination = (vt500-bumper, adxl-axle-up)

## 7.2 Vision Floor

Table 7.5 summarizes the setup for this experiment. (This is repeated from table 4.3.) We train on the set of previous locales (given in table 7.1), and then test on the current locale, *bramble1-1*. We train a multiclass model with the same 7 terrain classes that we used for the proprioception: grass, grass-leaves, pavement, soft vegetation, bramble, bushes, and trees. We test on the subset of classes at the current locale, *bramble1-1*: grass-leaves, pavement, bramble and treeBig.

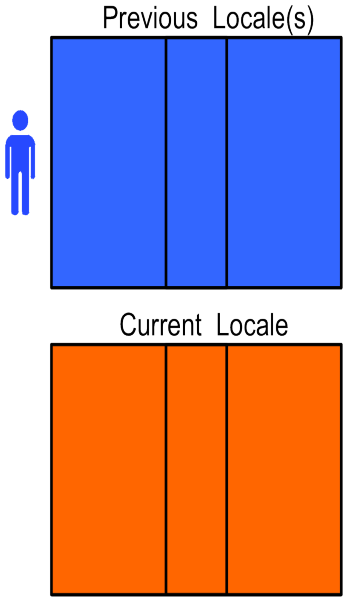
<ul style="list-style-type: none"> <li>• data source: camera images</li> <li>• data point format: local image patches centered around each pixel <ul style="list-style-type: none"> <li>▪ the subset of pixels used are those onto which Wobblers points are projected</li> </ul> </li> <li>• multiclass model: classes A, B, C, D (in reality, 7 classes)</li> <li>• training data <ul style="list-style-type: none"> <li>▪ labels for training: human (painted onto pixels in image)</li> <li>▪ location: previous locales, whole image</li> </ul> </li> <li>• test data <ul style="list-style-type: none"> <li>▪ labels for evaluation: human (painted onto pixels in image)</li> <li>▪ location: current locale, whole image</li> </ul> </li> </ul>	 <p>The diagram illustrates the data source and processing. On the left, a blue human icon stands next to two rows of colored rectangles. The top row, labeled 'Previous Locale(s)', consists of three blue rectangles. The bottom row, labeled 'Current Locale', consists of three orange rectangles. This represents the transition from training data (previous locales) to test data (current locale).</p>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Table 7.5: Vision Floor Overview

Table 7.6 lists the number of training and test data points for this experiment. The number of training data points is the union of data points for each class from the set of previous locales. For the test locale, there is only a subset of classes that show up, so *NA* is listed as the number of data points for the other classes. We limit the training data set for each class to the number of data points in the smallest class, whether ground or above-ground. We do not have to raise the threshold to the number of data points in the smallest ground class as we did for the proprioception experiments. The number of data points in the above-ground classes are now much larger than for the proprioception experiments because we are working with Wobblers points and pixels instead of short time windows. There are many Wobblers points that fall within the rectangular prism of a short time window. Refer to section 6.1 for a description of turning the exteroceptive data points into feature vectors, and then to section 6.2 for a description of training the feature vectors with a multiclass classifier.



Class	Training Previous Locales	Test Current Locale, <i>bramble1-1</i>
grass	62257	NA
grass-leaves	20625	21602
pavement	20890	14591
vegSoft	15300	NA
bramble	9032	4056
bush	13106	NA
treeBig	2481	690

Table 7.6: Number of Training and Test Data Points for Vision Floor Final Results

Table 7.7 presents the confusion matrix for the test predictions. These are predictions on the data points from the current locale, *bramble1-1*. The columns are ground-truth human labels on the test data, coming from the Gimp-painted labels. The rows are the predictions. The classes that show up at locale *bramble1-1* have non-zero columns in the confusion matrix.

		Actual Label						
		grass	grass-leaves	pavement	vegSoft	bramble	bush	treeBig
Predicted	grass	0	1010	17	0	1	0	0
	grass-leaves	0	11806	1427	0	382	0	76
	pavement	0	17	11072	0	497	0	0
	vegSoft	0	185	0	0	0	0	0
	bramble	0	1304	1775	0	1445	0	28
	bush	0	7204	0	0	1	0	4
	treeBig	0	76	300	0	1730	0	582

Table 7.7: Vision Floor Benchmark Confusion Matrix for Test Data (locale *bramble1-1*). Refer to appendix 9.15, 9.16, 9.17, 9.18, 9.19, 9.20, 9.21, 9.22 for training images, and then images 9.26, 9.27, 9.28, 9.29, 9.30, 9.31, 9.32, 9.33 for the human labels on that data. Refer to appendix 9.23, 9.24, 9.25 for test images, and then images 9.34, 9.35, 9.36 for test predictions on that data.

### 7.3 Vision Ceiling

Table 7.8 summarizes the setup for this experiment. (This is repeated from table 4.4.) We train on the subset of classes that show up inside the robot’s path at the current locale: grass-leaves, pavement, and bramble. We test on the subset of classes that show up outside the robot’s path at the current locale: grass-leaves, pavement, bramble, and treeBig. Note that because there is no training data for treeBig, it will never get predicted. This will result in misclassifications for this class. This is an error due to limitations in our experiments, and we discuss this in more detail in section 7.5.

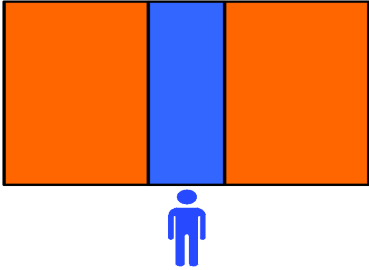
<ul style="list-style-type: none"> <li>• data source: camera images</li> <li>• data point format: local image patches centered around each pixel <ul style="list-style-type: none"> <li>▪ the subset of pixels used are those onto which Wobbler points are projected</li> </ul> </li> <li>• multiclass model: set of classes that actually show up in current locale: A, B, C</li> <li>• training data <ul style="list-style-type: none"> <li>▪ labels for training: human (painted onto pixels in image)</li> <li>▪ location: current locale, inside path</li> </ul> </li> <li>• test data <ul style="list-style-type: none"> <li>▪ labels for evaluation: human (painted onto pixels in image)</li> <li>▪ location: current locale, outside path</li> </ul> </li> </ul>	<p>Current Locale</p> 
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------

Table 7.8: Vision Ceiling Overview

Table 7.9 lists the number of training and test data points for this experiment. The number of training data points is the number of Wobbler points that fall inside the robot’s path. These are the data points that are contained within the the rectangular prisms of the robot’s footprints, and their class labels are determined by the labels of the footprints. Refer to section 4.1 for a discussion on registering exteroceptive data points to robot footprints. Note that we are using the human labels on the footprints, since this is the ceiling experiment. The number of test data points is the number of Wobbler points outside the robot’s path. These are labeled from the human Gimp labels. Note that there are zero training data points for the treeBig class, because it does not exist inside the robot’s path. When training, we limit the data points by the smallest class in the training set, whether ground or above-ground. Refer to section 6.1 for a description of turning the exteroceptive data points into feature vectors, and then to section 6.2 for a description of training the feature vectors with a multiclass classifier.

Class	Training Current Locale, bramble1-1 Inside Robot’s Path	Test Current Locale, bramble1-1 Outside Robot’s Path
grass-leaves	4612	16328
pavement	2315	10429
bramble	1140	3401
treeBig	0	690

Table 7.9: Number of Training and Test Data Points for Vision Ceiling Final Results

Table 7.10 shows the confusion matrix for the test predictions. These are predictions on the data points outside the robot’s path from the current locale, *bramble1-1*. The columns are ground-truth human labels on the test data coming from the Gimp-painted labels. The rows are the predictions. The four classes that show up in the test data have non-zero columns. The three classes that show up in the training data have non-zero rows.

		Actual Label						
		grass	grass-leaves	pavement	vegSoft	bramble	bush	treeBig
Predicted	grass	0	0	0	0	0	0	0
	grass-leaves	0	14451	135	0	74	0	70
	pavement	0	573	9973	0	438	0	0
	vegSoft	0	0	0	0	0	0	0
	bramble	0	1304	321	0	2889	0	620
	bush	0	0	0	0	0	0	0
	treeBig	0	0	0	0	0	0	0

Table 7.10: Vision Ceiling Benchmark Confusion Matrix for Test Data (locale *bramble1-1*, outside robot’s path). Refer to appendix 9.23, 9.24, 9.25 for bare images. Refer to appendix images 9.49, 9.50, 9.51 for human labels inside the robot’s path. Refer to appendix images 9.52, 9.53, 9.54 for visual test predictions outside the robot’s path.

## 7.4 Self-Supervision

Table 7.11 summarizes the setup for this experiment. (This is repeated from table 4.2.) We train on the subset of classes that are predicted by the proprioception to show up inside the robot’s path at the current locale: grass-leaves, pavement, and bramble. We test on the subset of classes that show up outside the robot’s path at the current locale: grass-leaves, pavement, bramble, and treeBig.

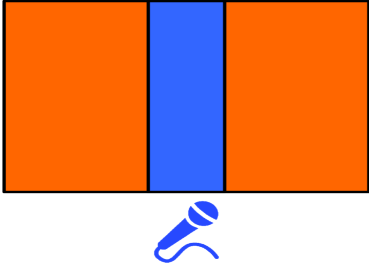
<ul style="list-style-type: none"> <li>• data source: camera images</li> <li>• data point format: local image patches centered around each pixel <ul style="list-style-type: none"> <li>▪ the subset of pixels used are those onto which Wobbler points are projected</li> </ul> </li> <li>• multiclass model: set of classes from proprioceptive predictions at current locale: A, B, C</li> <li>• training data <ul style="list-style-type: none"> <li>▪ labels for training: proprioceptive predictions mapped onto Wobbler points and then onto image patches</li> <li>▪ location: current locale, inside path</li> </ul> </li> <li>• test data <ul style="list-style-type: none"> <li>▪ labels for evaluation: human (painted onto pixels in image)</li> <li>▪ location: current locale, outside path</li> </ul> </li> </ul>	<div style="text-align: center;"> <p>Current Locale</p>  </div>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------

Table 7.11: Self-Supervision Overview

Table 7.12 lists the number of training and test data points for this experiment. The number of training data points is the number of Wobbler points that fall inside the robot’s path. These are the data points that are contained within the the rectangular prisms of the robot’s footprints, and their class labels are determined by the labels of the footprints. Refer to section 4.1 for a discussion on registering exteroceptive data points to robot footprints. Note that we are using the proprioceptively-predicted labels on the footprints. These predictions are coming from the output of the proprioception experiment in section 7.1. These predictions are smoothed with a mode filter before being used in this step. This helps with smoothing out noise and improves the accuracy of these predictions. (We discuss this in section 5.5.) The number of test data points is the number of Wobbler points outside of the robot’s path. These are labeled from the human Gimp labels. When training, we limit the data points by the smallest class, whether ground or above-ground. Refer to section 6.1 for a description of turning the exteroceptive data points into feature vectors, and then to section 6.2 for a description of training the feature vectors with a multiclass classifier.

Class	Training Current Locale, bramble1-1 Inside Robot’s Path Proprioceptively-Predicted Labels	Test Current Locale, bramble1-1 Outside Robot’s Path Human Labels
grass-leaves	2461	16328
pavement	1636	10429
bramble	1339	3401
treeBig	0	690

Table 7.12: Number of Training and Test Data Points for Self-Supervision Final Results

Table 7.12 shows the confusion matrix for the test predictions. These are predictions on the data points outside the robot’s path from the current locale, *bramble1-1*. The columns are ground-truth human labels on the test data coming from the Gimp-painted labels. The rows are the predictions. The four classes that show up in the test data have non-zero columns. The three classes that show up in the training data have non-zero rows.

		Actual Label						
		grass	grass-leaves	pavement	vegSoft	bramble	bush	treeBig
Predicted	grass	0	0	0	0	0	0	0
	grass-leaves	0	13178	154	0	102	0	94
	pavement	0	230	9992	0	709	0	0
	vegSoft	0	0	0	0	0	0	0
	bramble	0	2920	283	0	2590	0	596
	bush	0	0	0	0	0	0	0
	treeBig	0	0	0	0	0	0	0

Table 7.13: Self-Supervision Confusion Matrix for Test Data (locale *bramble1-1*, outside robot’s path), using proprioception with (vt500-bumper, adxl-axle-up) sensors. Refer to appendix 9.23, 9.24, 9.25 for bare images. Refer to appendix images 9.43, 9.44, 9.45 for proprioceptively predicted labels inside the robot’s path. Refer to appendix images 9.46, 9.47, 9.48 for visual test predictions outside the robot’s path.

## 7.5 Analysis

Table 7.14 lists the recall rates for the four main experiments. The left column lists the three out of the seven terrain classes that show up inside the robot’s path at the current locale, *bramble1-1*. The next four columns are the recall values for each for the four experiments. The bottom row shows the average recall across all classes for each experiment. These recall rates can be extracted from the confusion matrices in tables 5.14, 7.7, 7.13 and 7.10 for each of the four experiments.

Class	Proprio	Vision floor	Self-sup	Vision ceiling
grass-leaves	0.71	0.55	0.81	0.89
pavement	0.72	0.76	0.96	0.96
bramble	1.00	0.36	0.76	0.85
average	0.81	0.56	0.84	0.90

Table 7.14: Recall Comparison. Each column is generated from confusion matrices 5.14, 7.7, 7.13, 7.10, respectively.

Section 2.4 introduced the three dimensions of the thesis problem. Section 4.4 laid out how we will prove that we addressed each dimension. To decrease human effort, we said we would show that supervised proprioception outperforms supervised exteroception when training on one set of locales and testing on another. The hypothesis is that exteroception is not consistent across locales, but proprioception is. We prove this here by comparing the proprioception column and the vision floor column. We can see that the average of the proprioception is better than the average of the vision floor. However, the proprioception column is only showing accuracies for the test predictions on data inside the robot’s path, whereas the vision floor column is showing accuracies for test predictions on a larger set of data outside the robot’s path. This brings us to the next dimension of decreasing robot effort.

To decrease robot effort, we said that we would leverage the consistency of the proprioception into the exteroceptive field of view, making accurate predictions on the larger exteroceptive set of data. We said we would show that self-supervision outperforms supervised exteroception when training on one set of locales and testing on another. We prove this here by comparing the self-supervision column and the vision floor column. We can see that the average of the self-supervision is better than the average of the vision floor. This is testing on the larger data set outside the robot’s path.

Furthermore, we can compare the vision ceiling column to the self-supervision column. The average of the vision ceiling is better than the average of the self-supervision. Here we have replaced the proprioceptively-predicted labels on the training data with human labels. This is the best we can expect the model to do given:

1. the ability of the training data to represent all classes
2. the ability of the feature space to separate the classes
3. the ability of the classifier to recognize the separations in the feature space

The vision ceiling allows the performance of the self-supervision to be relatively evaluated against this upper benchmark. We can see that the self-supervision performance is quite close to the ceiling.

Table 7.15 lists the normalized precision values for the same four main experiments. The meanings of each row and column are analogous to the recall rates in table 7.14. Again, the values can be extracted from the confusion matrices in tables 5.14, 7.7, 7.13 and 7.10 for each of the four experiments. The average in the proprioception column is better than the average of the vision floor column. So this speaks again to proving the hypothesis that proprioception is more consistent than exteroception across locales. This

is tackling the second dimension of the thesis problem, human effort.

<b>Class</b>	<b>Proprio</b>	<b>Vision floor</b>	<b>Self-sup</b>	<b>Vision ceiling</b>
grass-leaves	1.00	0.64	0.82	0.87
pavement	1.00	0.86	0.81	0.85
bramble	0.64	0.62	0.42	0.46
<b>average</b>	<b>0.88</b>	<b>0.71</b>	<b>0.68</b>	<b>0.73</b>

Table 7.15: Normalized Precision Comparison. Each column is generated from confusion matrices 5.14, 7.7, 7.13, 7.10, respectively.

We do not see improvement, however, when comparing the self-supervision column and the vision floor column of the normalized precisions in table 7.15. This is directly related to the fact that the tree terrain class does not show up inside the robot’s path, thereby limiting the self-supervision classifier’s ability to correctly predict the tree labels in the test data. Looking at the self-supervision confusion matrix in 7.13, we can see that the data points in the tree column are mostly confused as bramble. They have no way of being correctly predicted as treeBig. This, in turn, decreases the normalized precision for the bramble class, which we can see here as the value 0.42 in table 7.15. This brings down the average quite a bit. The vision floor has tree data in its training set from the previous locales, so it does not suffer from this problem. Note that the recall for the self-supervision is not affected by the tree data, because the recall rates are calculated separately for each column.

The normalized precision for the vision ceiling also performs poorly for the same reasons as the self-supervision. Looking at the vision ceiling confusion matrix in 7.10, we can see again that the data points in the tree column are mostly confused as bramble. So this again decreases the normalized precision for the bramble class to a value of 0.46 in table 7.15. This again brings down the average quite a bit. Above, we listed three constraints on the ceiling performance. In this case, it is the first constraint that is limiting the ceiling: the ability of the training data to represent all classes.

Table 7.16 lists the normalized precision values, but now ignoring the tree column in the confusion matrices for the self-supervision and vision ceiling. (The proprioception and vision floor columns are repeated from the previous table.) The self-supervision and vision ceiling columns are being generated from the same two confusion matrices, 7.13 and 7.10, respectively. But now the treeBig column is being ignored. When ignoring the tree data, we can see the same trend as we did in the recall table: the self-supervision is outperforming the vision floor. The vision ceiling, in turn, is higher than the self-supervision. The self-supervision once again is quite close to the vision ceiling. We consider the lack of tree data in the training set to be an edge case. It is a limitation directly related to our experimental setup. We discuss in the future work section (8.2) how active learning might be used to make sure that all terrain classes at the current locale are sampled in the training data.

<b>Class</b>	<b>Proprio</b>	<b>Vision floor</b>	<b>Self-sup</b>	<b>Vision ceiling</b>
grass-leaves	1.00	0.64	0.95	0.96
pavement	1.00	0.86	0.81	0.85
bramble	0.64	0.62	0.79	0.88
<b>average</b>	<b>0.88</b>	<b>0.71</b>	<b>0.85</b>	<b>0.90</b>

Table 7.16: Normalized Precision Comparison Without Tree. Each column is generated from confusion matrices 5.14, 7.7, 7.13, 7.10, respectively. The last two columns for self-supervision and vision ceiling ignore the tree data in the confusion matrix, since there is no training data for the tree class.

We examine normalized precision rather than precision because it normalizes out the effect that the number of data points for each class has on the precision values. There is less tree data because it is an above-ground class. If we were to look at the precision values instead of the normalized precision values, the problems that the tree data causes would not be as evident. Often, the hazardous classes, which are the most precious to predict, have the least number of data points, and we don't want these results to get drowned out by the ground terrain. This, in general, is a reason why we might choose to look at normalized precision instead of precision. But in analyzing the tree edge case here, we are showing another reason why it is a good idea to look at normalized precision: so that we can see trends in edge cases such as these.



# Chapter 8

## Conclusion

In summary, we have proven the ability of our algorithm to address both the human effort and the robot effort dimensions of the thesis problem. We have shown this for both the recall and normalized precision without the tree edge case. We have demonstrated that proprioception is more invariant than exteroception. Therefore, proprioception can be used to improve exteroception from locale to locale, while still preserving the ability of the exteroception to classify at a range without having to interact with that terrain directly.

### 8.1 Contributions

<b>We are the first to build a rich semantic classifier that includes multiple ground and above ground terrain classes, using only medium levels of human and robot effort.</b>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

In table 2.3, we summarized how prior work has addressed the three dimensions of our thesis problem. These three dimensions are: semantic richness of the terrain classification, human effort, and robot effort. The table ended by showing how we would extend the prior work, which we succeeded in doing. To deliver on the first dimension, increasing the semantic richness of the classifier, we built a seven-class multiclass classifier that included both ground and above-ground terrain types. The ground types included grass, grass-leaves and pavement. The above-ground types included soft vegetation, bushes, bramble, and trees. To deliver on the second dimension, decreasing robot effort, we built a supervised proprioceptive multiclass classifier that can generalize well between different environments. This allowed the robot to train on a small subset of locales, and then test on new locales with high accuracy. To deliver on the third dimension, decreasing human effort, we implemented a self-supervised system using proprioception to teach exteroception. The proprioception allows for better model generalization across locales. The exteroception, when retrained by the proprioception at each new locale, can transmit that better model generalization into its larger field of view. The retrained exteroception can then make better predictions, which can be registered onto a larger map of the terrain. We tested these assumptions against clear benchmarks, with data collection on a fully integrated robotic vehicle.

**We implemented a two-step framework using proprioception to teach exteroception, and we are the first to show the benefit of this two-step framework for rich, semantic terrain classification.**

Traditional self-supervision uses a direct framework, where one sensing modality provides direct labels to the second sensing modality. Our scheme uses a two-step framework, where there is an auxiliary preprocessing step, or first step, that uses supervised learning to teach the first sensing modality on a subset of data. We are using proprioception as the first sensing modality and exteroception as the second sensing modality. This two-step framework using proprioception to teach exteroception has been explored to some extent by other researchers, but it is not common, and we implemented the richest semantic multiclass terrain classifier with this approach. Previous work has only built ground terrain classifiers, and we extended this work to multiple ground and above-ground terrain classes.

**In our ICRA paper [22], we were the first to use sound for mobile robot terrain classification, with the ability to classify a semantically rich set of classes.**

At the time of writing our ICRA paper, Ojeda et al. [25] were the only other ones to use sound at all for mobile robot terrain classification, and they only looked at ground terrain types. They were successful mostly with the grass terrain type using the sound modality. We built a multiclass classifier using only sound that could distinguish between multiple ground classes and one above-ground class. (Others built off our work in supervised acoustics-based terrain classification, but sticking to ground classes [10], [42], [41], [35]). We explored various feature extraction techniques coming out of acoustics literature, and we demonstrated that both our *gianna and shape* and *log bins* feature vectors worked well. We demonstrated how these feature vectors could be successfully trained in a multiclass classifier, and we showed that binary SVM nodes within a one-vs-one multiclass scheme provided good results. We also showed in Bagnell et al. [6] that these techniques easily generalize to interactive sensing for robotic manipulator applications. In general, we have tested these techniques in very challenging outdoor environments with messy data, and so these techniques are positioned to perform very well in cleaner indoor environments.

**We are the first to use air microphones as part of the proprioception teacher.**

We are the first (and only) ones to use air microphones as a sensing modality in the proprioceptive teaching suite within a self-supervision framework for mobile robot terrain classification. Other researchers have used vibration, and we extended this work by using just air microphones, as well as a combination of vibration and air microphones.

**We determined the best pair of sound and vibration sensors for measuring ground and above-ground interaction.**

We extended our early acoustics work by exploring the use of different sound and vibration sensors, both alone and in combination with each other. We explored 19 different sound and vibrations signals, coming from condenser air microphones, electret air microphones, contact microphones, accelerometers, and piezoelectric vibration sensors. We explored the placement of these sensors on the axle of the vehicle for measuring ground terrain interactions and sensors within the vehicle's bumper for measuring above-ground terrain

interactions. We learned a combination of these sensors through an automated validation process. We determined that a combination of an air microphone near the bumper and a z-axis acceleration signal near the axle was the best combination. We demonstrated that the feature extraction and classification methods that we explored in our early work with sound could extend well to different sound sensors, as well as to the sensors in the vibration modality. We also demonstrated that these feature extraction and learning techniques could extend to a combination of the signals, by concatenating the features together. And we also demonstrated that these techniques could extend across robotic platforms from the Gator platform to the LAGR platform.

**Our exteroceptive classification provides evidence that the Variational Auto Encoder (VAE) is a powerful tool for mobile robot visual terrain classification.**

We implemented a Variational Auto Encoder (VAE) in order to learn the feature space from unsupervised camera data. This is leveraging a state of the art deep learning technique in computer vision. We showed this could work for the Pixim HDR cameras on our platform, which is a common camera used on field robotic vehicles. We showed that the VAE feature choice worked much better than Gabor features, a traditional choice used for texture classification. We showed this working on data that consisted of local image patches associated with points from a point cloud of (x,y,z) locations in a map of the surrounding terrain. Sometimes VAEs are used within an image segmentation scheme, and we show here a working implementation of using them on just local image patches. Using them on just local image patches allows them to be tagged to specific 3d points in the terrain map, thereby allowing them to be used within mapping applications. Separate from whether or not the end application is mapping, such mapping is necessary for the registration within our self-supervised framework. The association of local image patches with 3d points is what allowed the exteroceptive training data to be labeled from the proprioceptive domain. We demonstrated the use of VAE feature vectors as input to a supervised multiclass classifier built from SVM's within a one-vs-one multiclass scheme. This is using an unsupervised deep learning neural network along with a final supervised SVM layer. This in itself is an interesting implementation, and it is a good example of leveraging the strength of unsupervised deep learning for computer vision along with the ability of SVM's to perform well on small labeled datasets.

**We created self-supervision architecture that will allow the research community to understand these concepts, communicate about them, and build off them in the future.**

Because the two-step self-supervised framework that we used is not common, it has not been fleshed out on a conceptual level that concretely, which can lead to a great deal of confusion when discussing this research. We concretely described both the direct and two-step frameworks with diagrams in section 2.2. We then described how we apply the two-step framework to our mobile robotics system in section 4.2 with diagrams to explain how the two step framework relates to the data registration necessary for such an implementation. Finally, we used these diagrams to define our four main experiments in section 4.3, which allowed our results to be compared to floor and ceiling benchmarks in a clear fashion.

We feel that these conceptual diagrams are in themselves a contribution to the research community for understanding and communicating about these concepts.

**We developed a process for projecting time-series labels into the exteroceptive field of view.**

The process of projecting human time-series labels into the exteroceptive field of view is in itself a technical contribution that could aid researchers in labeling exteroceptive data, even if just working on supervised classification problems. In section 4.1, we described the process of labeling sequences in the time series data, turning short time windows into rectangular footprints, tagging the 3d point clouds with labels from those footprints, and then projecting these labels onto the image plane. For a particular data trial, the robot makes a path through the locale. Only a few start and end timestamps need to be tagged in order to provide time-series labels. For instance, if there are three terrain-interaction events, then the human labeler need only mark six points in a time-series signal (the beginning and end of each event.) This information can then be used to project labels onto the point cloud stream, and in turn the camera stream, for that entire trial. For a human to label all of that point cloud data or all of those images would be very labor intensive. So even if the application is for a supervised exteroceptive classifier, having the proprioceptive signals from the robot-terrain interaction just as a human hand-labeling tool allows for exteroceptive data to be labeled much more easily. The process we have described for this data registration is in itself a contribution for easing the process of human-labeling. Semi-supervised methods for integrating human labels with proprioceptive predictions could be further explored, and we discuss this in the future work section (8.2).

**We collected a dataset with many sensing modalities across diverse terrain for use by other researchers. We also developed a robotic platform for future data collection.**

We generated a very unique dataset which could be used in the future by the robotics, machine learning, and computer vision communities. This data includes 19 sound and vibration signals, image streams from HDR color cameras, 3d point clouds with near-uniform density, and other proprioceptive signals coming from the low-level controller. All of this data has the information to be time synchronized and spatially registered in post-processing. We collected the data over a variety of locations in outdoor off-road environments, including Schenley Park, Raccoon Creek State Park, and the Gascola and Taylor CMU test sites. The data collection trials contain many robot-terrain interaction events with both ground and above-ground terrain classes. The trials also contain camera and 3d point clouds from multiple distances away from target terrain classes, with the robot stopping every two meters to collect 3d point clouds with no motion blur. We retrofitted the LAGR platform with all of the sensors discussed above. We put a great deal of systems work into this platform, and it is still available for future robotic use.

## 8.2 Future Work

We feel that the highest impact opportunity for extending off our work would be to incorporate our self-supervised framework into a hierarchical scheme. Figure 8.1 shows again the two-step framework that we implemented. In this framework, the classification is set up as a large multiclass problem, where a single multiclass classifier (one for each of the two steps in the framework) is responsible for estimating all of the semantic classes.

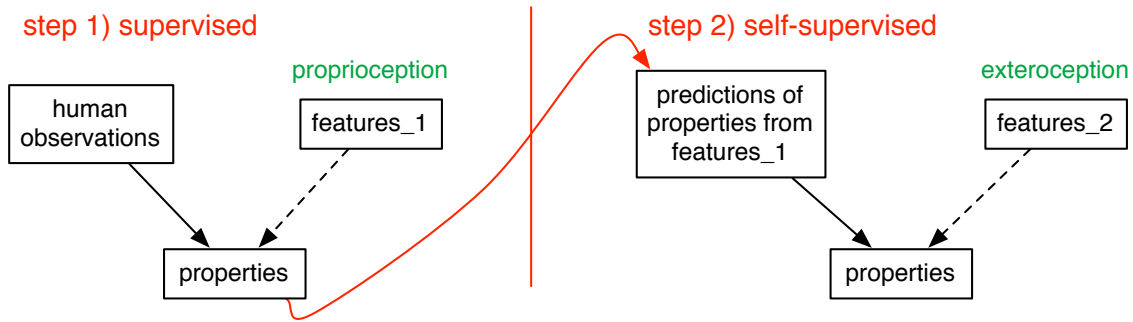


Figure 8.1: Our current two-step self-supervised framework

Instead of considering all of the semantic classes as part of a flat set, a taxonomy of terrain categories could be considered, where subsets of terrain classes are grouped together. These hierarchical groupings would allow the classification task to be broken up into stages, where each stage is only responsible for a smaller set of classes. Figure 8.2 shows an example of how a large set of terrain classes might be organized into such a hierarchy.

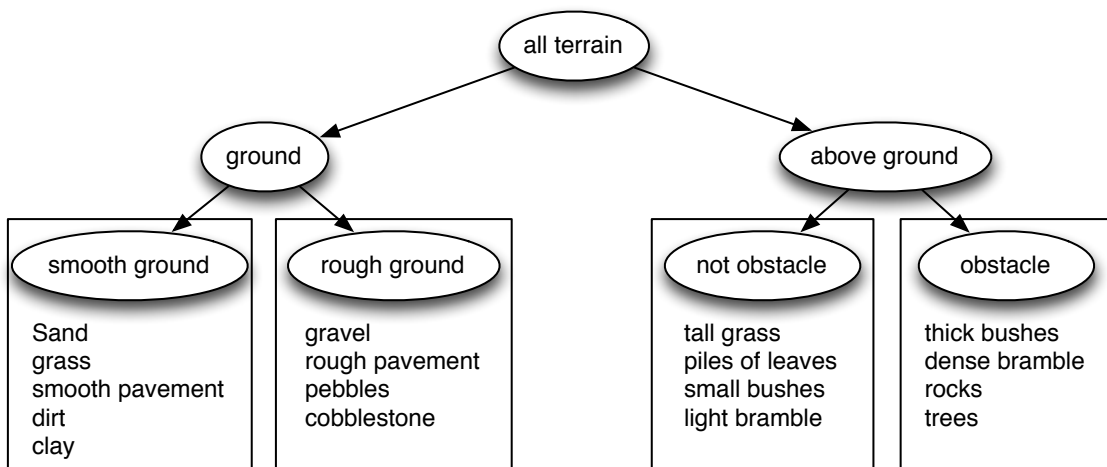


Figure 8.2: A hierarchical scheme to semantic labeling of terrain classes

Each upper split of this tree is a binary classification problem, and then the leaf nodes are separate multiclass classification problems. In this example, the binary classification problems above the leaves can be modeled *directly* without human labels. For the first level of the tree, ground versus above ground, a model could be built from height values provided by the 3d point clouds coming from range data. A plane could be fit to the ground. Along the normal to that plane, a threshold could be chosen that corresponds to a significant height above the ground plane. For distinguishing obstacle versus not obstacle on the second level of the tree, a model could be built from bumper hits, telling the robot whether it has hit something that is traversable or not. This could also be determined with velocity information instead of bumpers: if the velocity of the robot is zero, then the robot has hit an obstacle. Note that this is a measure of the *terrain compliance*, similar to the direct properties being learned in prior work as listed in table 2.2. For distinguishing smooth ground versus rough ground, one could use a measure of *terrain roughness*, also a direct property from table 2.2. Similar to these authors, one could use a one-dimensional measure of the energy in each short window (integral of the z-axis amplitude of an accelerometer) and pick a reasonable threshold to split the ground types into *rough* and *not rough* categories.

Once these direct measurements have been used to reach the leaf nodes of the tree, the two-step framework could be used for each leaf. The reader might visualize the entire framework in figure 8.1 shrunken down and stuffed into each leaf node. The direct measurements (height, velocity, bumpers hits, energy of z-acceleration) would be used to separate the data points into subsets (leaf nodes) and then each subset of the data would be handled by the two-step framework. Note that the human labels needed for the first step of the framework would not have to be repeated; they would simply be labels for each of the data points as they were before, and then a subset of the data points would be fed into each leaf. Each model would only be responsible for a subset of the classes, allowing them to be easier multiclass problems.

Our current two-step implementation uses *learned* property predictions from sound and vibration features. These learned predictions would still be used in each leaf node as part of this hierarchical framework. But now the initial hierarchical step of breaking the entire set of data points into subsets would be handled by *direct* predictions coming from a different suite of proprioceptive sensors (bumpers, pose, energy of z-acceleration) as well as some straightforward exteroceptive data (height). This staged approach is our attempt to combine both direct and learned proprioceptive predictions. One can think of this combination as incorporating both the direct self-supervised framework from figure 2.1 and the two-step framework that we implemented (figure 8.1).

One problem that could arise is an overlap of terrain classes into multiple groups. For instance, pavement could be rough or smooth, depending on how recently the ground has been paved. Hence, in figure 8.2, we split pavement up into two separate semantic classes, smooth pavement and rough pavement. Research into what choices to make in the taxonomy process for semantic categorization is another area of future work. This falls into the realm of cognitive science and how semantic taxonomies should be organized.

In our current two-step framework, we have a suite of teacher measurements from proprioceptive data and student measurements from exteroceptive data. Now with the

hierarchical framework, the direct measurements at the upper levels of the tree are being added to the suite of teacher measurements. Note that *height* is coming from range data, which is exteroceptive. This starts to break down the assumption that proprioception is always the teacher. This leads to an exploration on how parts of the exteroception might be incorporated into the suite of teacher measurements. Using height from range data is just one example. Other exteroceptive measurements might also be used as part of the teacher suite. Our general assumption is that exteroceptive labels will not be robust to new environments, and therefore should not be used as the teacher. But perhaps *some* of the exteroceptive data *is* robust enough to be used on its own or in combination with the proprioceptive data. Experiments could be conducted to test this hypothesis.

Another variant on our current two-step framework in figure 8.1 could be to make the first step unsupervised instead of supervised. This is depicted in figure 8.3. The second self-supervised step would be the same as before. In the first step, there would be no human labels, and the proprioceptive features would be turned into property labels through unsupervised learning. There would still be a trivial human-labeling step to tag each cluster with a terrain class. This entire framework could again be visualized as shrunken down and stuffed into the leaf nodes of the hierarchical tree from figure 8.2. Using the tree to split the large multiclass problem into smaller multiclass problems would allow the unsupervised clustering to have more of a chance at success.

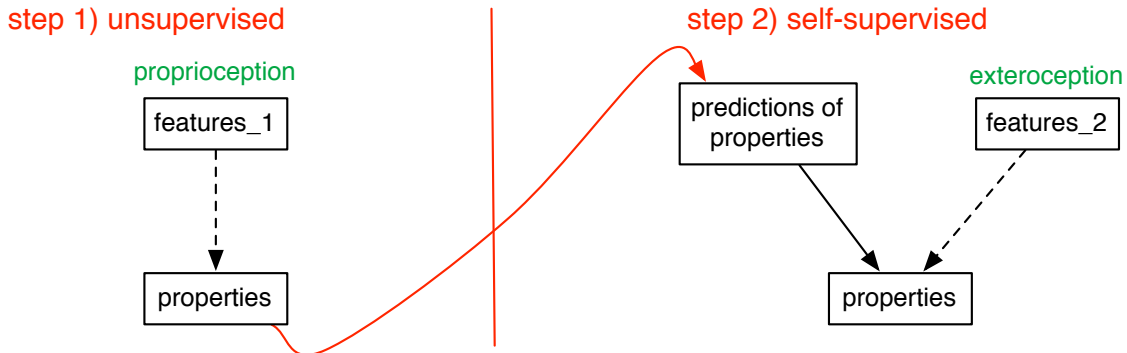


Figure 8.3: A variant to the two-step framework in figure 8.1. Here, the first step is unsupervised instead of supervised. There would still be a trivial step of tagging each unsupervised cluster with a human label.

Instead of a fully unsupervised first step, we might also experiment with *semi-supervised* techniques. For example, at each locale, we have a subset of the terrain classes. The data from this locale could be fed into an unsupervised clustering algorithm to separate the classes, and then have a human trivially tag each cluster with a label. Note that this is different than the fully unsupervised approach we introduced above because the human is tagging clusters for each new locale that is used for training the proprioception. Lower-dimensional features might be sufficient for distinguishing between the smaller set of classes at a particular locale, and the clustering algorithm might have a better chance of succeeding with these lower dimensional features. This clustering would be an alternative to the fully

supervised process of hand-labeling the starting and ending timestamps for each interaction event. This would be especially useful for above-ground terrain classes, where labeling the start and end of each interaction event has to be done more often and with more resolution. The semi-supervised data from each locale could then be combined with data from other locales into a full training dataset. This whole process would replace the first step from the two-step framework (figure 8.1). The semi-supervised scheme we are describing can be summarized as follows:

1. Each signal file would be treated as a separate classification problem on the subset of classes at that locale, which would be solved with unsupervised clustering on low-dimensional proprioceptive features.
2. The clusters would then be tagged with human labels.
3. The labeled data points from each file would be combined into a larger set with labeled data points from other files.
4. A supervised model would be learned from this training data, using higher-dimensional proprioceptive features.
5. The predictions from this model would then be used as the teacher in the second step of the two step framework to train the exteroceptive features.

In our current exteroception module, we use a VAE to learn the feature space for our exteroceptive features in an unsupervised fashion. Similarly, we might use a VAE on our proprioceptive signals to learn a feature space. The work we have done in extracting robust proprioceptive features would act as a nice benchmark for seeing if we get improvement with the VAE. This could also be useful in learning lower-dimensional features for semi-supervised labeling within each locale, as discussed above. In this case, we might retrain a VAE at each new locale in an unsupervised fashion.

In section 7.5 we discussed the fact that the tree class was not showing up in our data inside the robot’s path at the current test locale. However, the tree data did show up outside the robot’s path. This is because there were trees at the locale, but the robot never interacted with any of them. So when the proprioception was retraining the exteroception at this locale, it did not have tree as a class in its model. This was a limitation in the experimental setup of our data collection trials. However, this could also be an issue in a fully autonomous system where there is no human supervisor telling the robot what terrain to sample. In these cases, active learning approaches might be useful, where the robot makes decisions about what terrain it wants to sample, and then plans a path through the terrain accordingly. Unsupervised clustering on the exteroceptive data could be used to split the data up into groups, and without worrying about the terrain label for each group, the robot might try to sample some subset of each cluster. (Ott and Ramos [27] separated visual data into unsupervised clusters, tagging each cluster with a label from the bumper information about whether it was an obstacle or not.)

We learned the VAE features in an unsupervised fashion, but this is a separate step from the unsupervised clustering being discussed here. The image patch data points would be transformed into a feature space, perhaps through a VAE implementation. No matter what feature extraction technique is used, this unsupervised clustering step would happen



on the data once it is in the feature space. It might be beneficial to use a lower-dimensional feature vector for this clustering step, since we are clustering on the subset of the data at this locale. This is similar to what we discussed above for clustering the proprioceptive features in a lower-dimensional space for semi-supervised labeling. Furthermore, the VAE features could be relearned at each new locale, perhaps in a lower dimension, and then these relearned features could be fed into the clustering algorithm.

An area of robotics research that we are particularly passionate about is systems integration; namely, the aggregation of different sub-systems into a coordinated whole that is greater than the sum of its parts. In this thesis, we have brought together the sub-systems of proprioception and exteroception into a more advanced robotics perception system. By using self-supervision, we have implemented a sensor fusion system that leverages the advantages of each sensing modality. We have worked with both hardware and software to build a fully integrated robotic perception system. In doing so, we were able to make advances in interactive perception, going beyond the majority of robotic perception systems, which are passive in nature. We hope that these contributions can help future roboticists carve out systems integration research problems with confidence, allowing for sub-components that can elegantly interact with each other, and allowing for robots that can elegantly interact with the world.



# Chapter 9

## Appendix

### 9.1 Convolutional Variational Auto Encoder Images



Figure 9.1: 100 Input Image Patches of the Grass Class

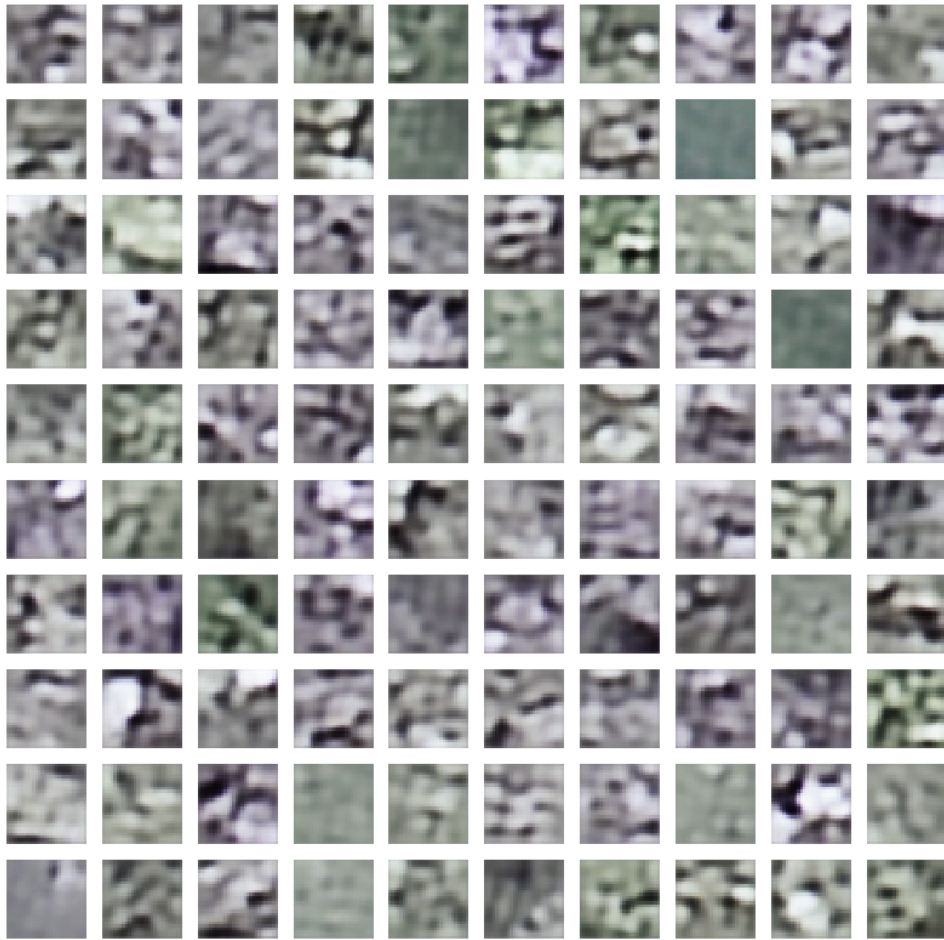


Figure 9.2: 100 Decoded Outputs of the Grass Class, corresponding to inputs in figure 9.1



Figure 9.3: 100 Input Image Patches of the Grass-leaves Class

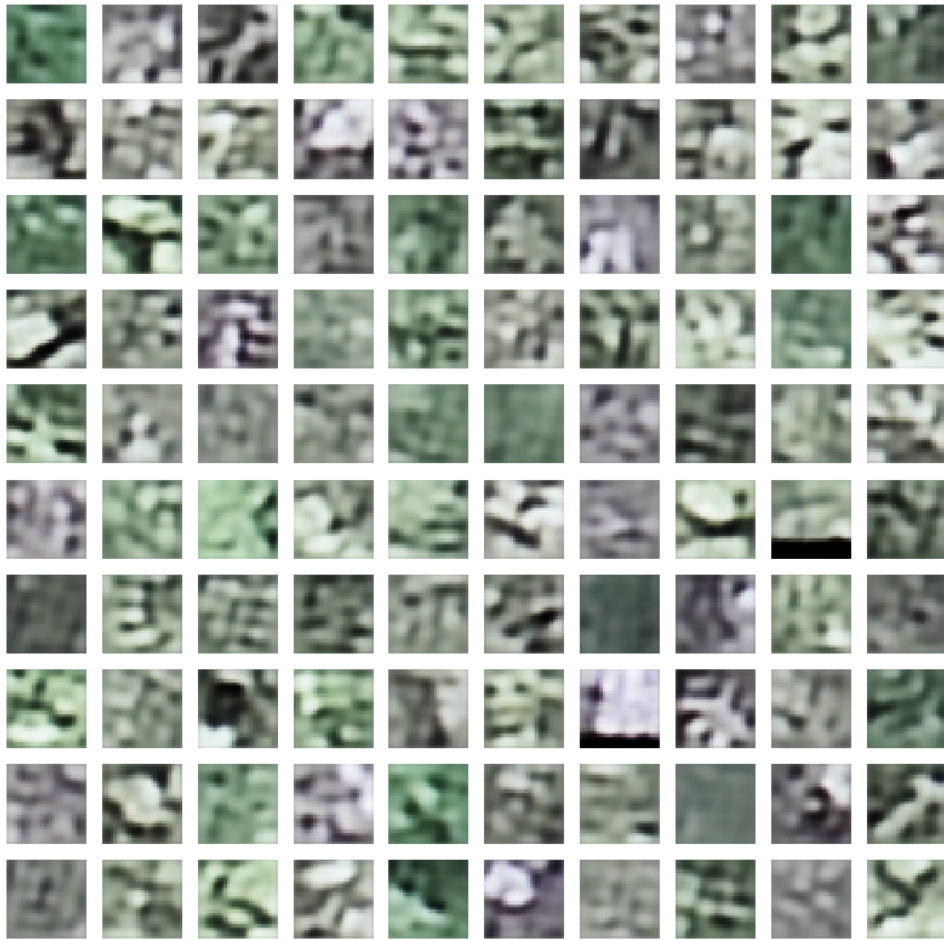


Figure 9.4: 100 Decoded Outputs of the Grass-leaves Class, corresponding to inputs in figure 9.3

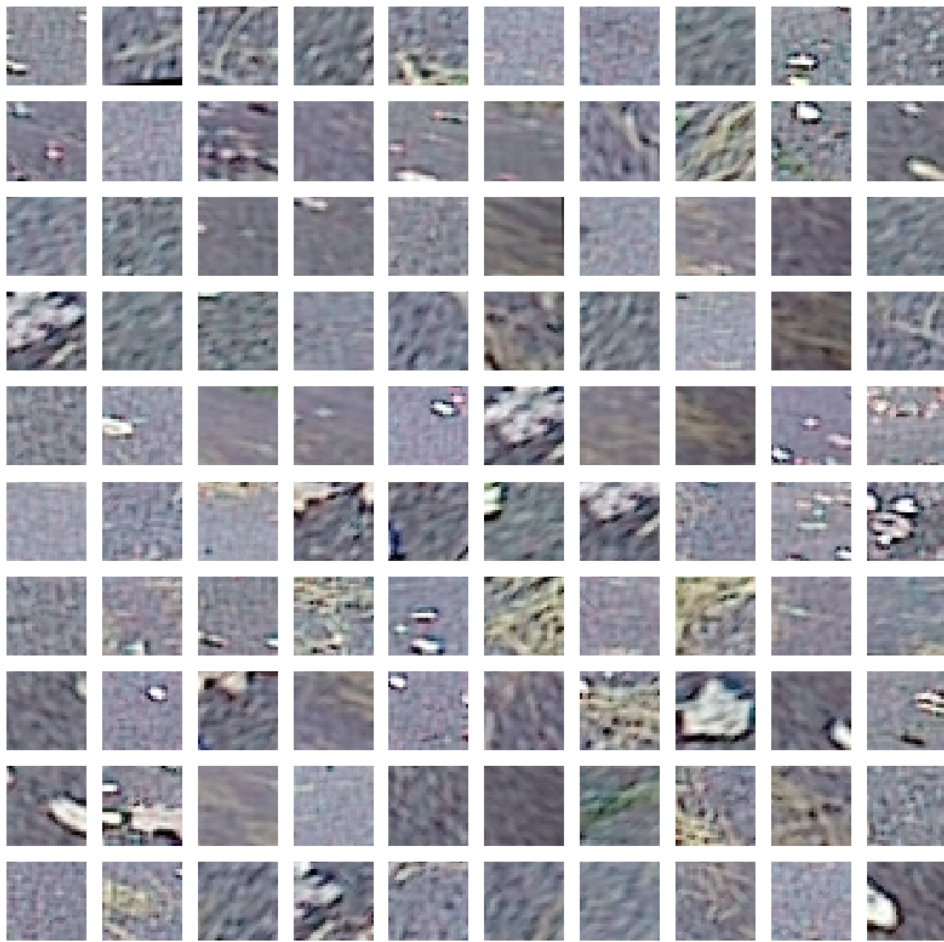


Figure 9.5: 100 Input Image Patches of the Pavement Class



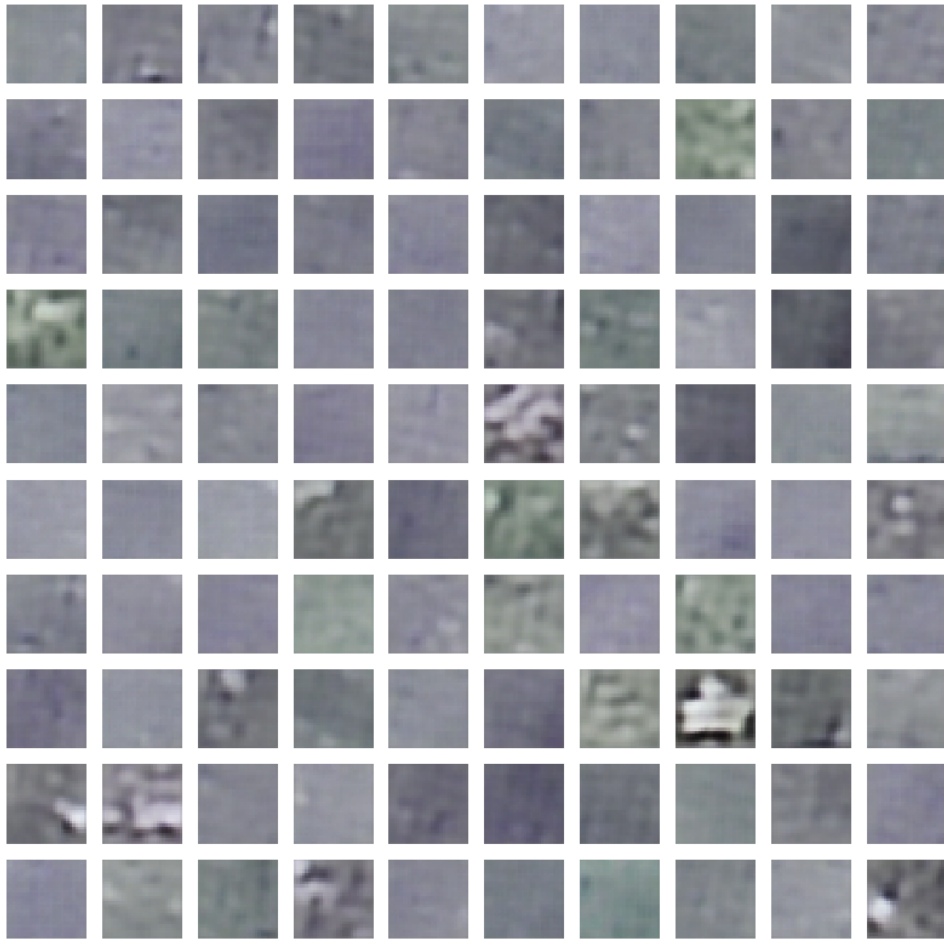


Figure 9.6: 100 Decoded Outputs of the Pavement Class, corresponding to inputs in figure 9.5



Figure 9.7: 100 Input Image Patches of the Soft Vegetation Class

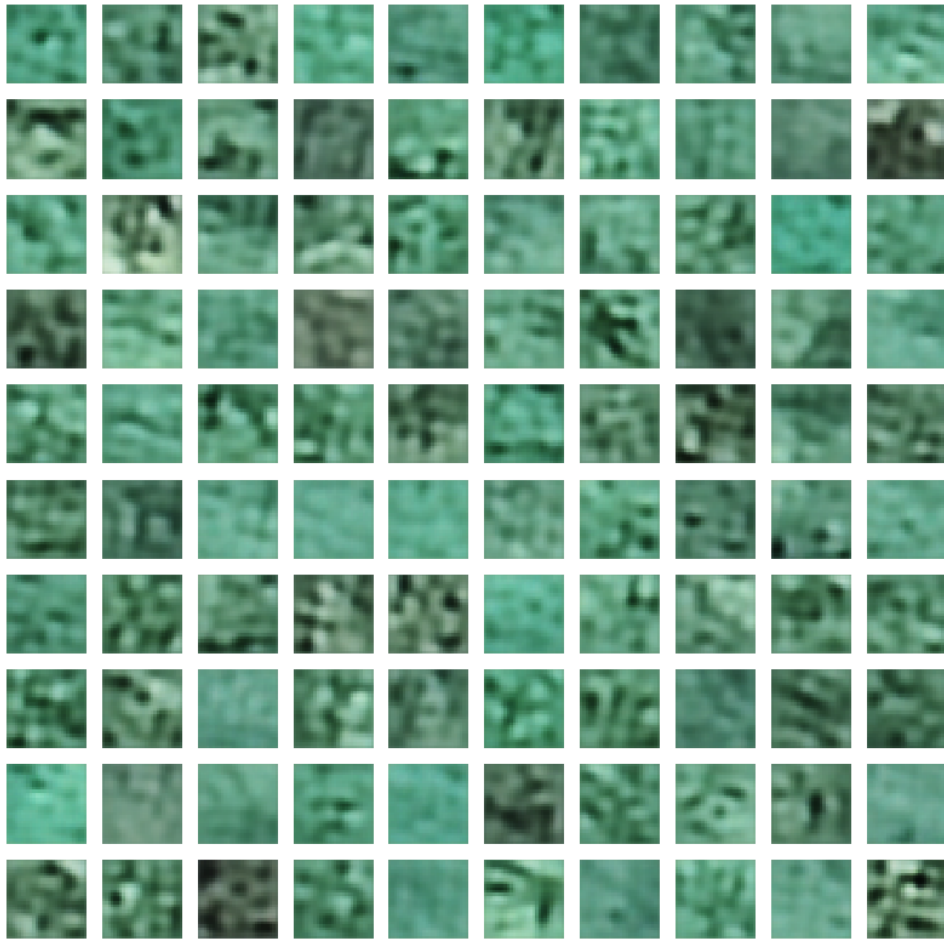


Figure 9.8: 100 Decoded Outputs of the Soft Vegetation Class, corresponding to inputs in figure 9.7



Figure 9.9: 100 Input Image Patches of the Bramble Class

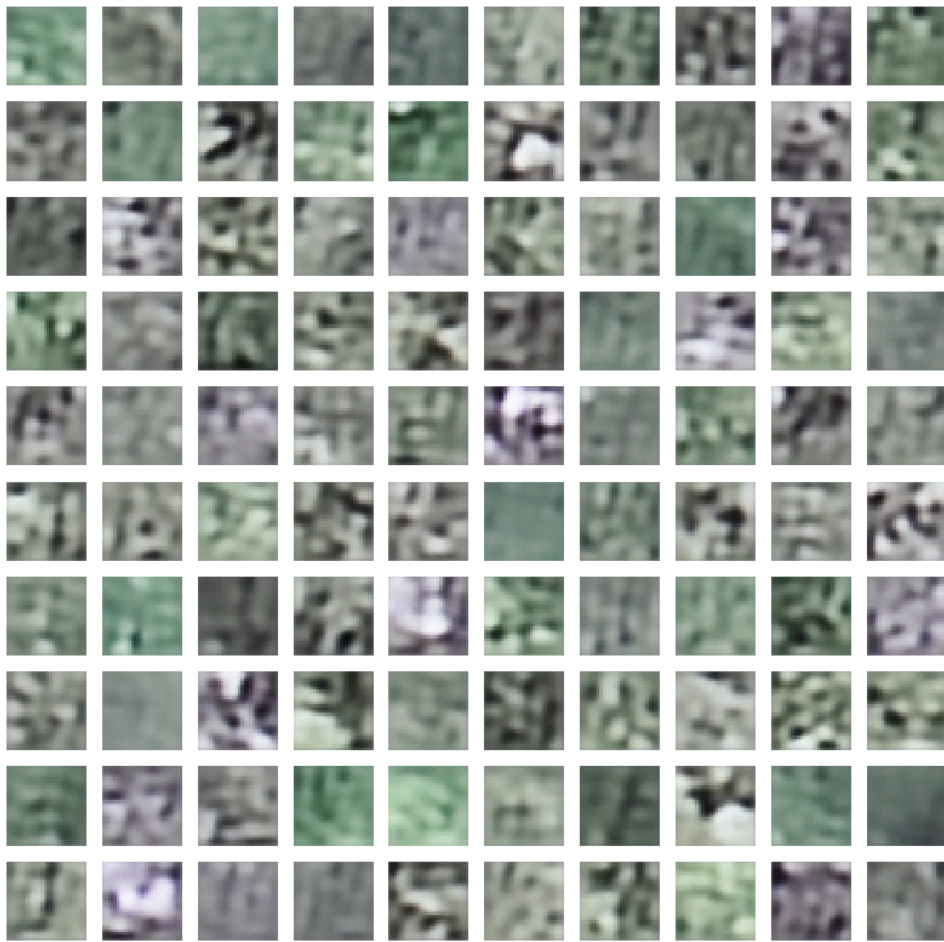


Figure 9.10: 100 Decoded Outputs of the Bramble Class, corresponding to inputs in figure 9.9

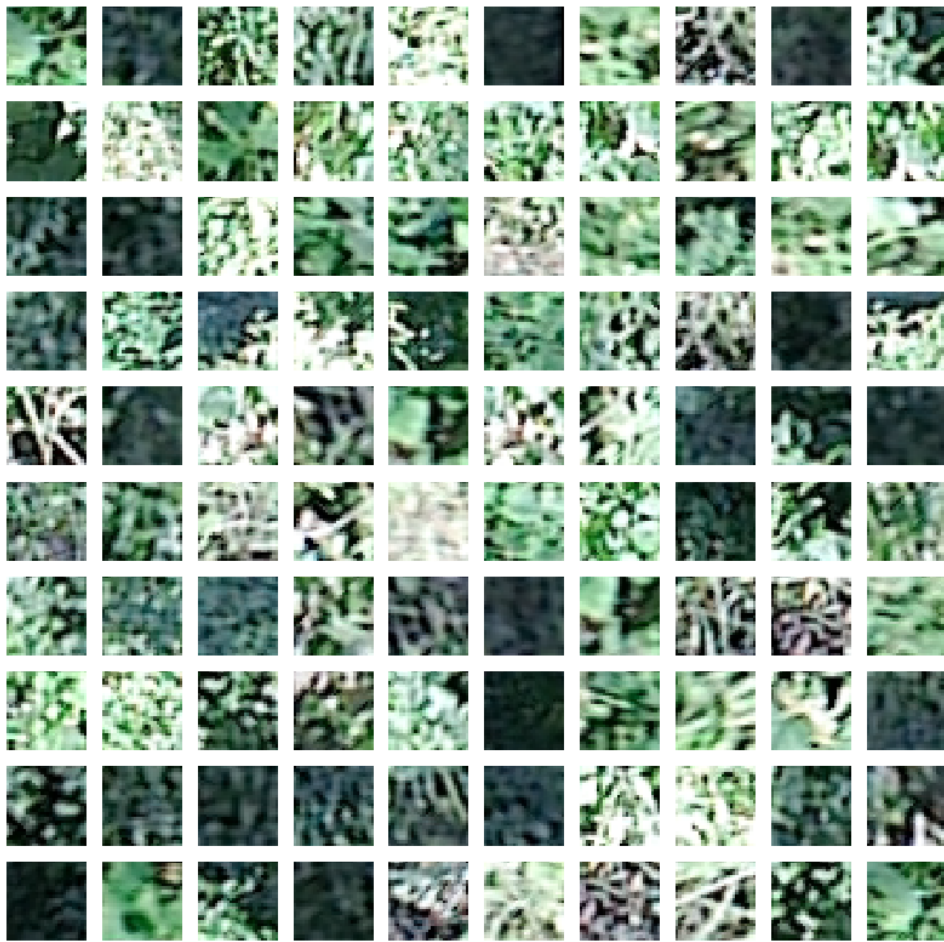


Figure 9.11: 100 Input Image Patches of the Bush Class

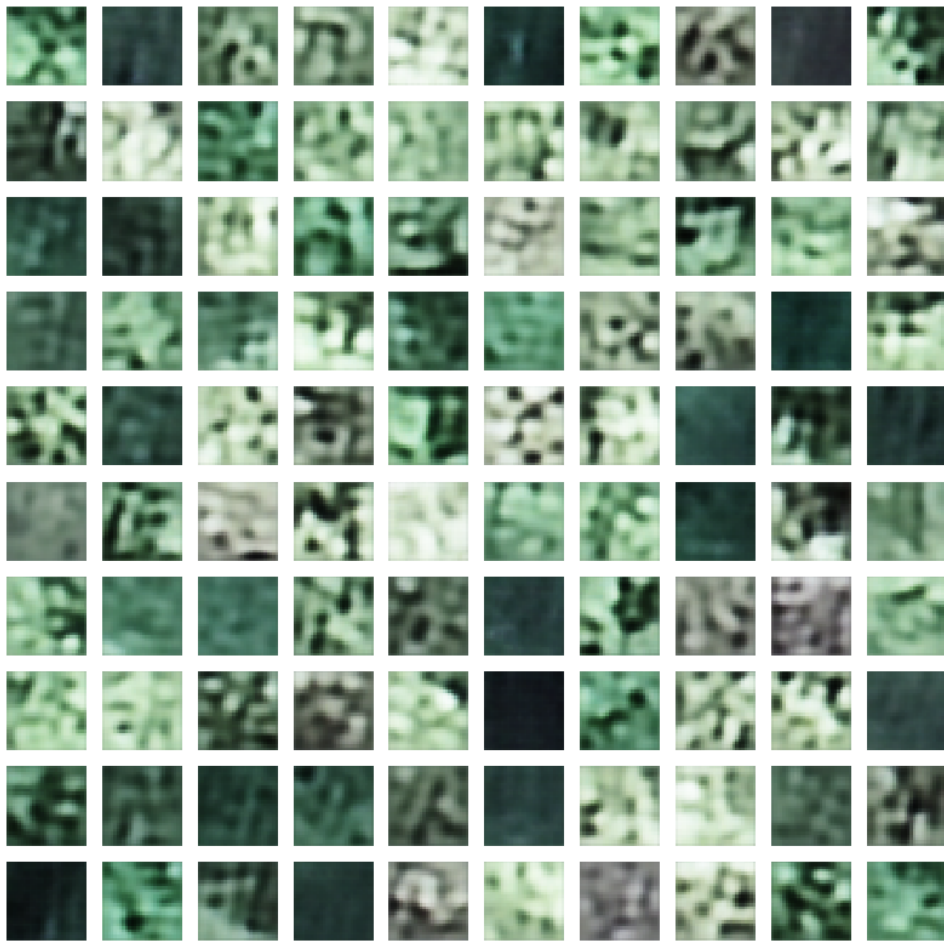


Figure 9.12: 100 Decoded Outputs of the Bush Class, corresponding to inputs in figure 9.11

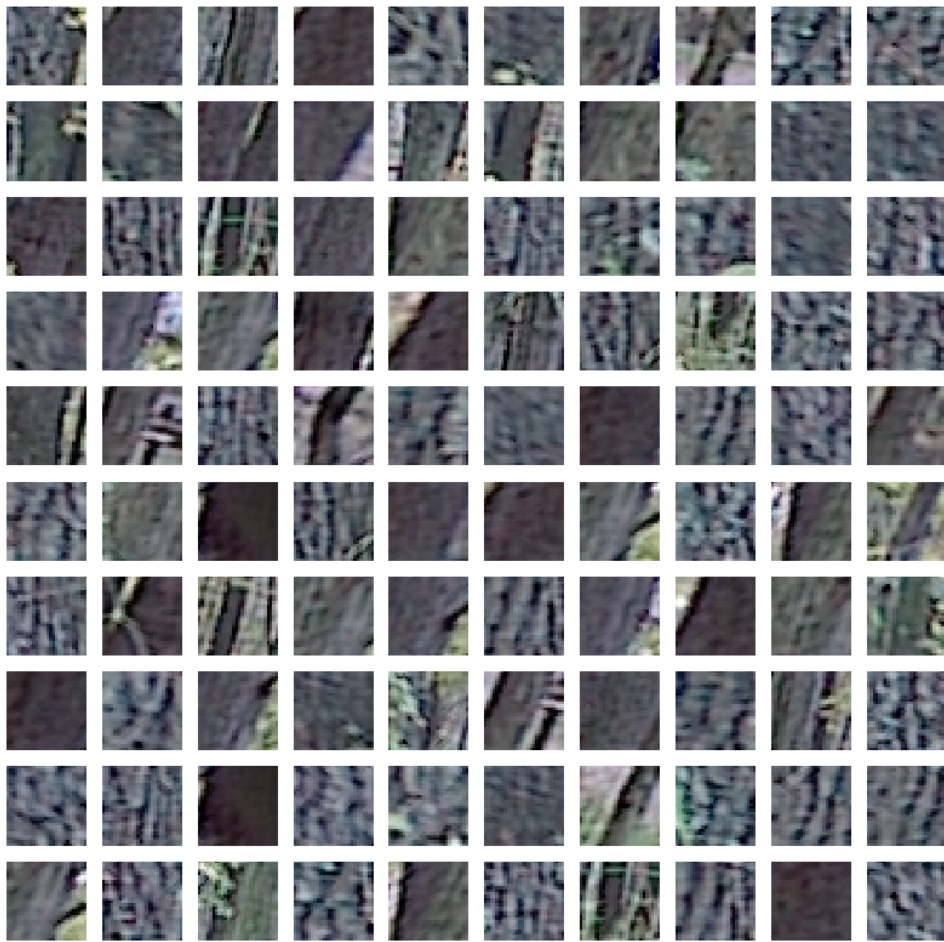


Figure 9.13: 100 Input Image Patches of the Tree Class



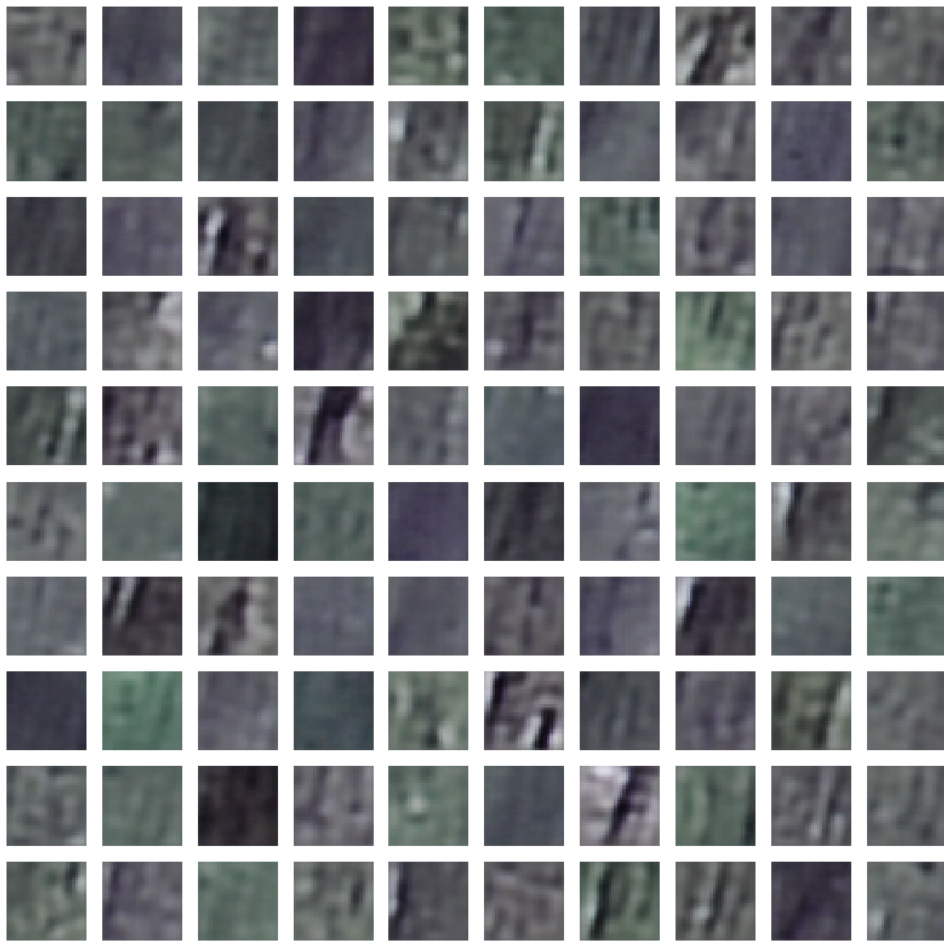
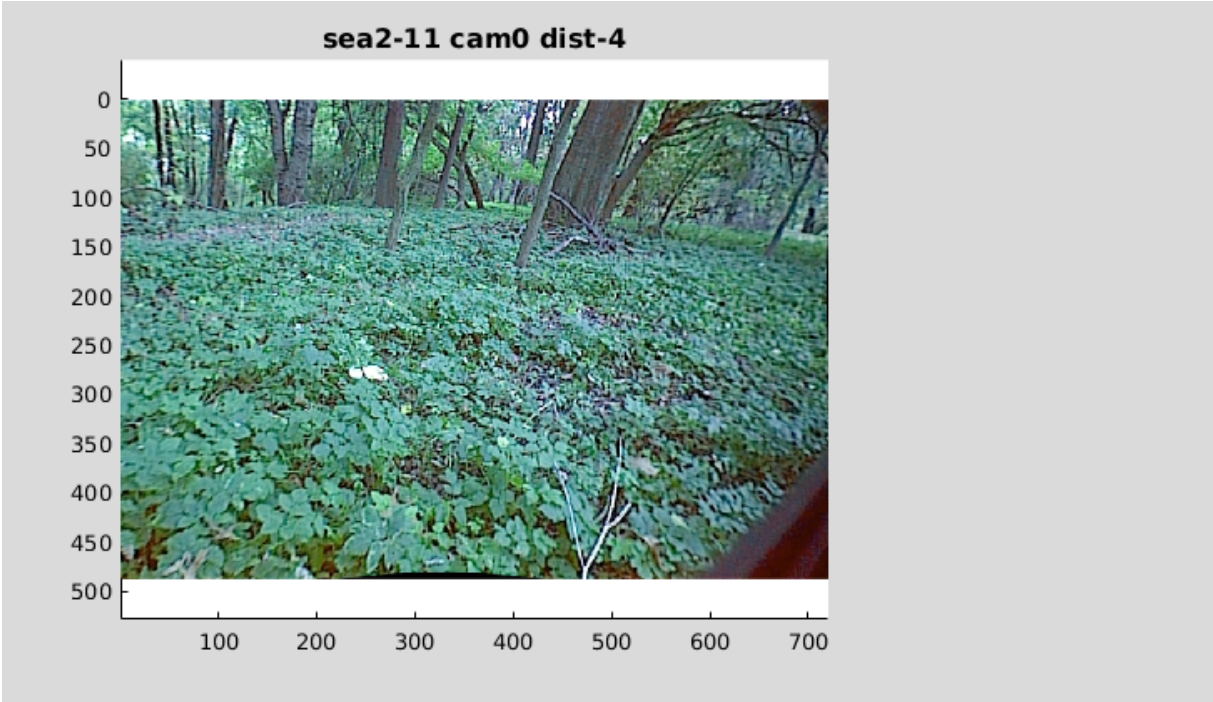


Figure 9.14: 100 Decoded Outputs of the Tree Class, corresponding to inputs in figure 9.13

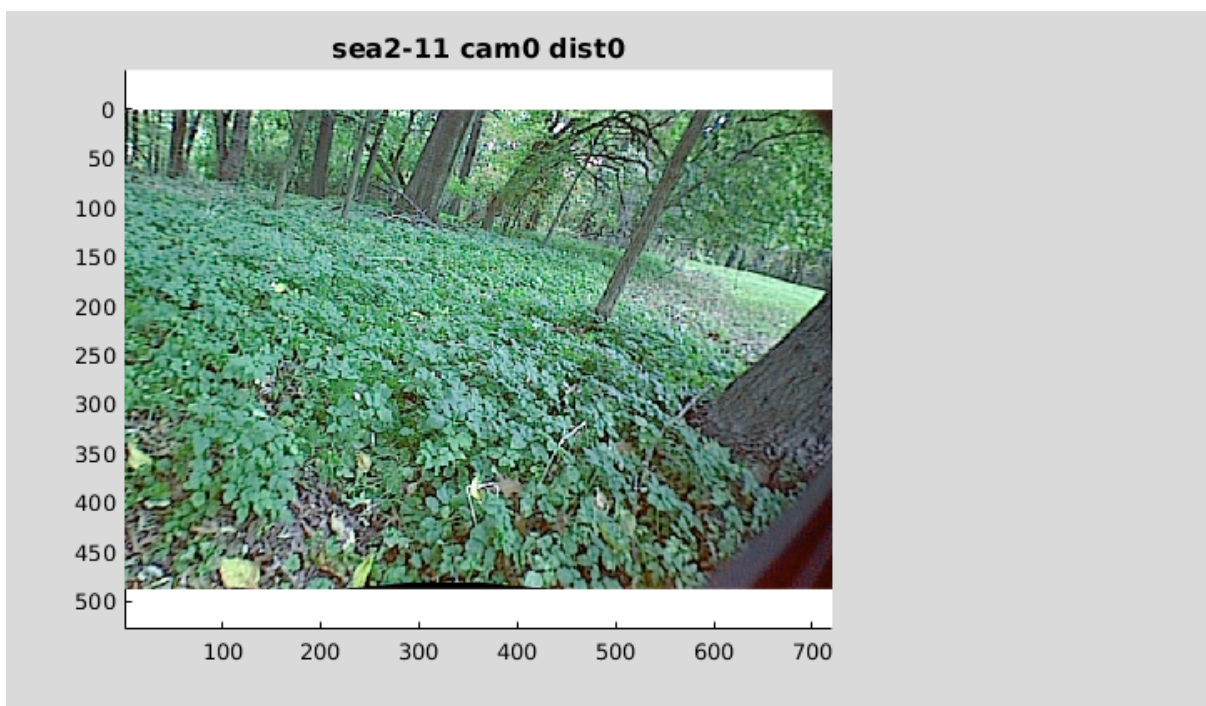
## 9.2 Self-Supervision Images

### 9.2.1 Pixim Images for Previous Locales (sea2-11, wood1-7, wood1-8, bush4-1, tree2-1)

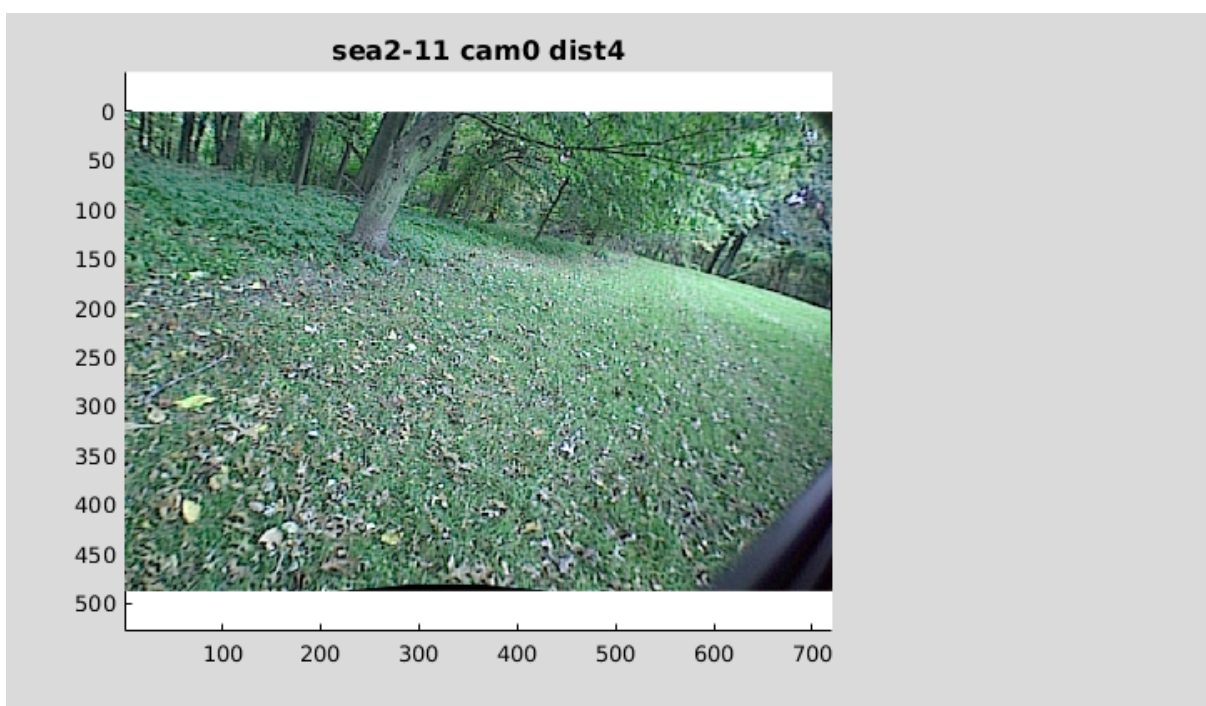


(a)

Figure 9.15: Pixim Images for Previous Locales (sea2-11, wood1-7, wood1-8, bush4-1, tree2-1), Image 1. This set of previous locales is used for training the vision floor benchmark. Refer to images 9.26, 9.27, 9.28, 9.29, 9.30, 9.31, 9.32, 9.33 for human labels on this data.

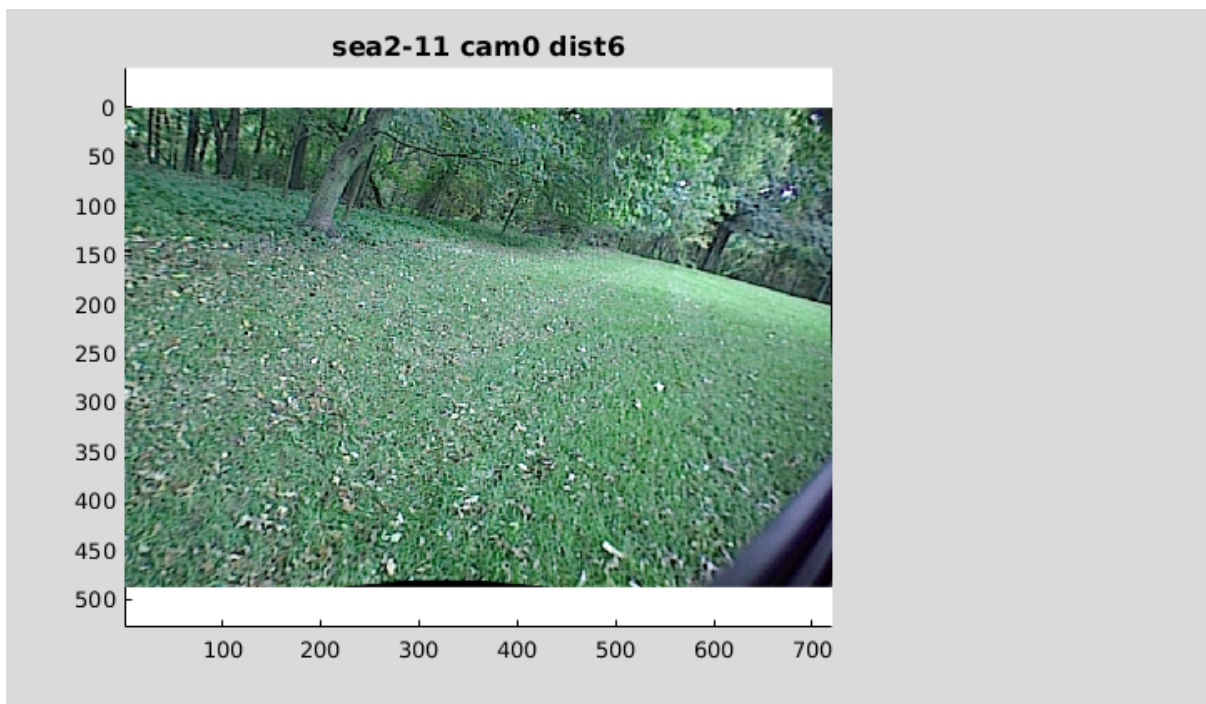


(a)

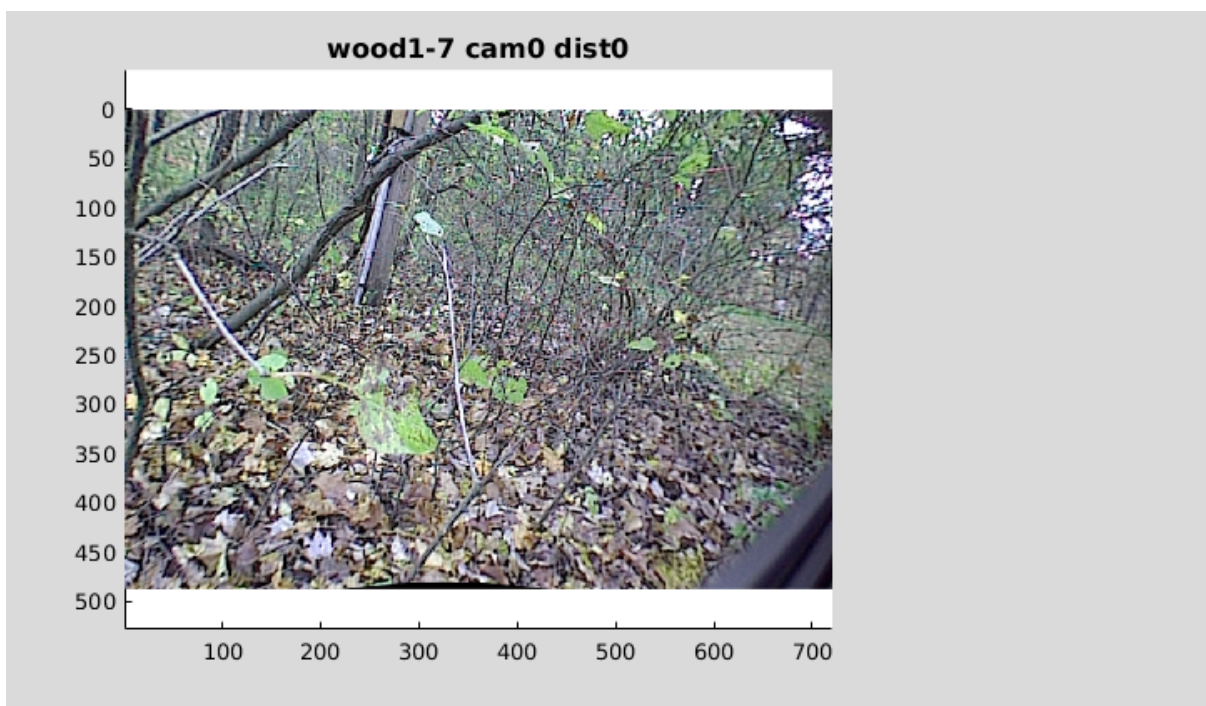


(b)

Figure 9.16: Pixim Images for Previous Locales (sea2-11, wood1-7, wood1-8, bush4-1, tree2-1), Continued, Images 2 - 3

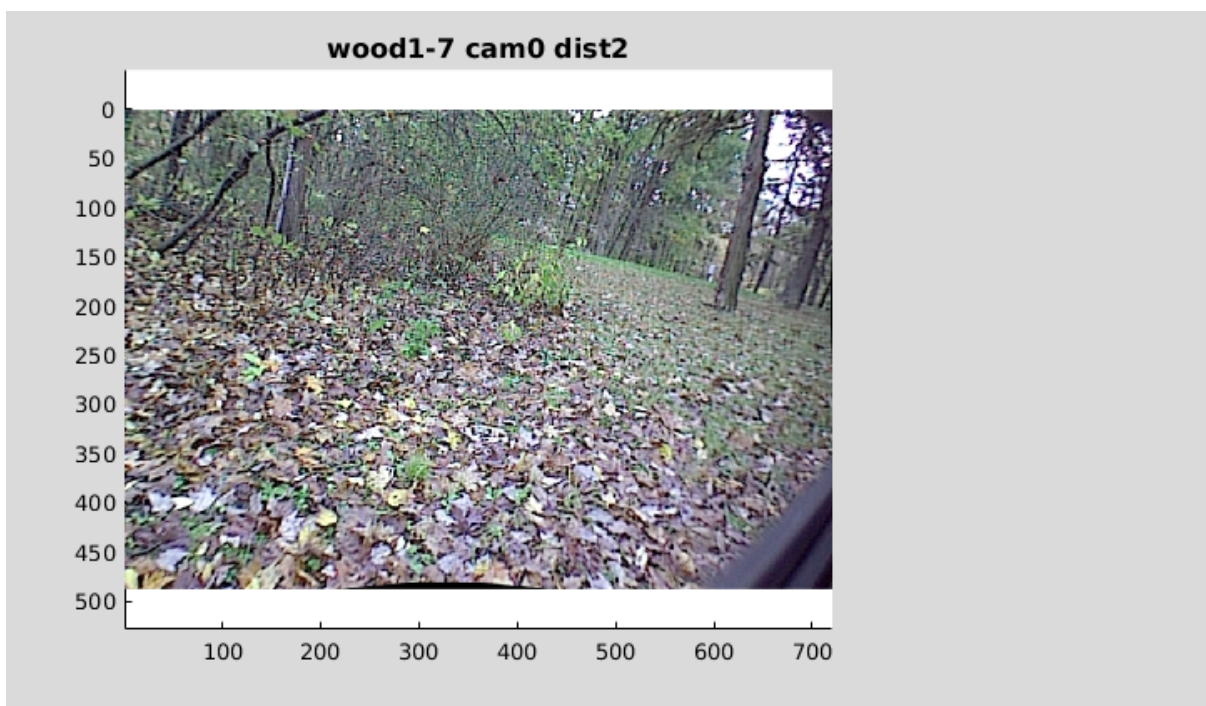


(a)

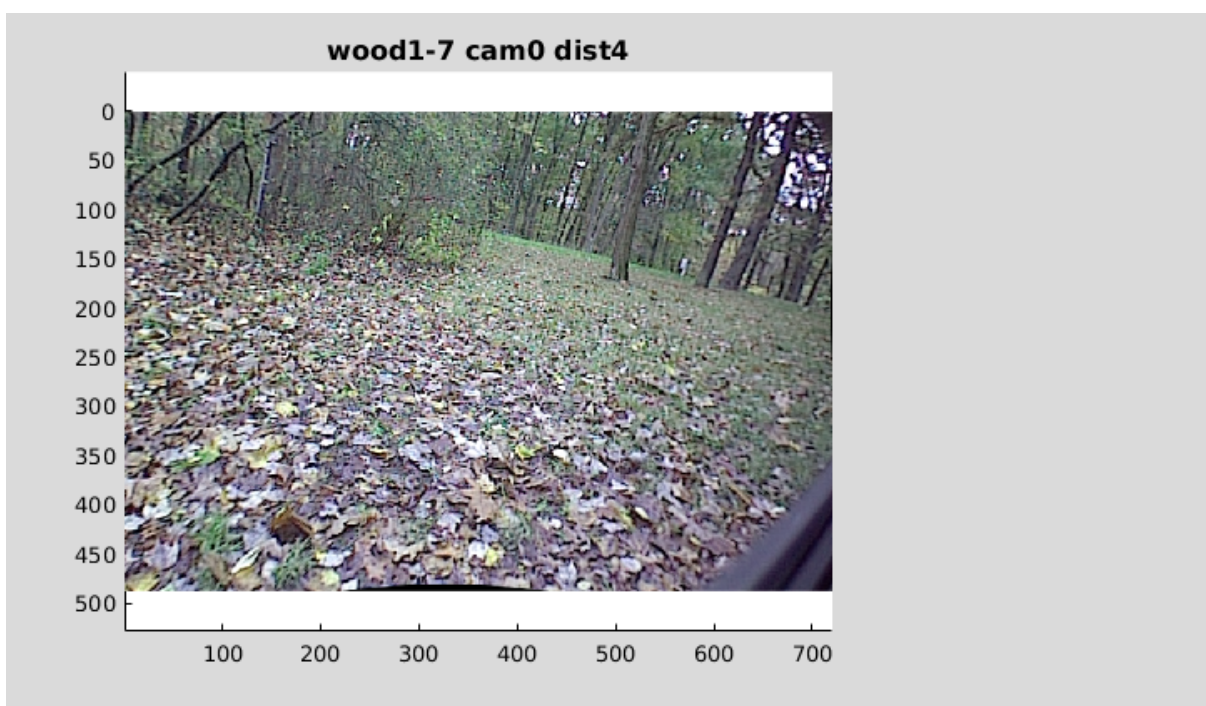


(b)

Figure 9.17: Pixim Images for Previous Locales (sea2-11, wood1-7, wood1-8, bush4-1, tree2-1), Continued, Images 4 - 5

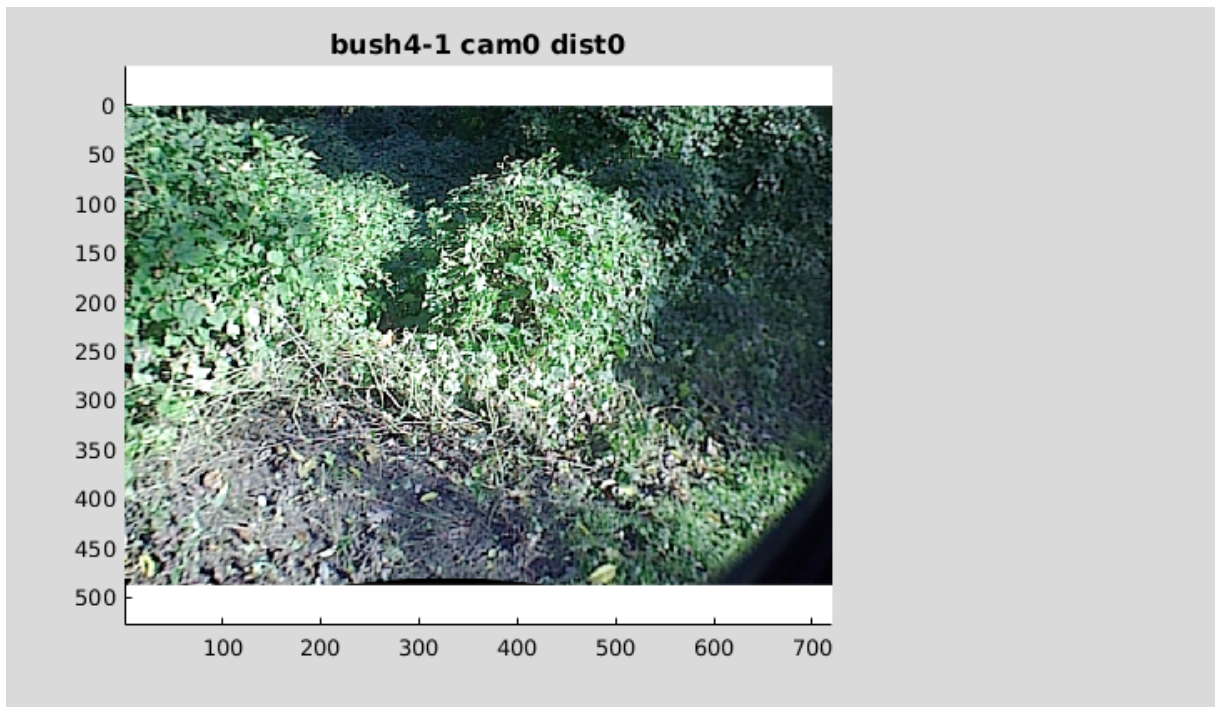


(a)

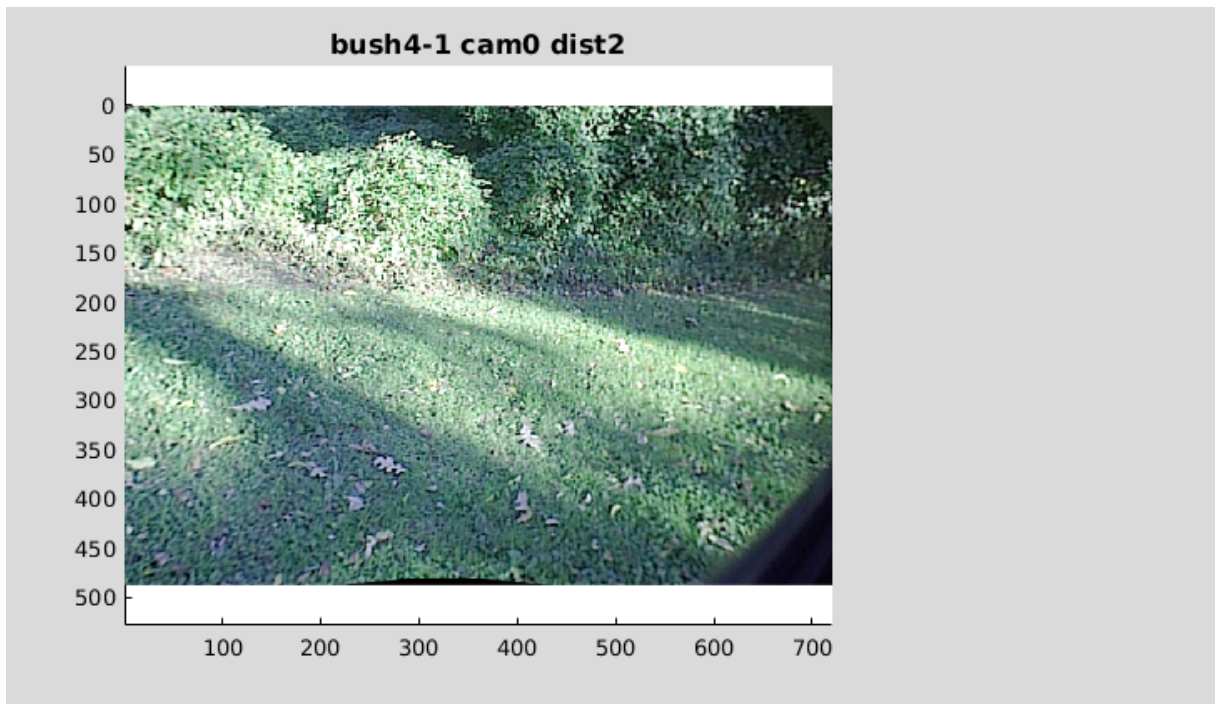


(b)

Figure 9.18: Pixim Images for Previous Locales (sea2-11, wood1-7, wood1-8, bush4-1, tree2-1), Continued, Images 6 - 7

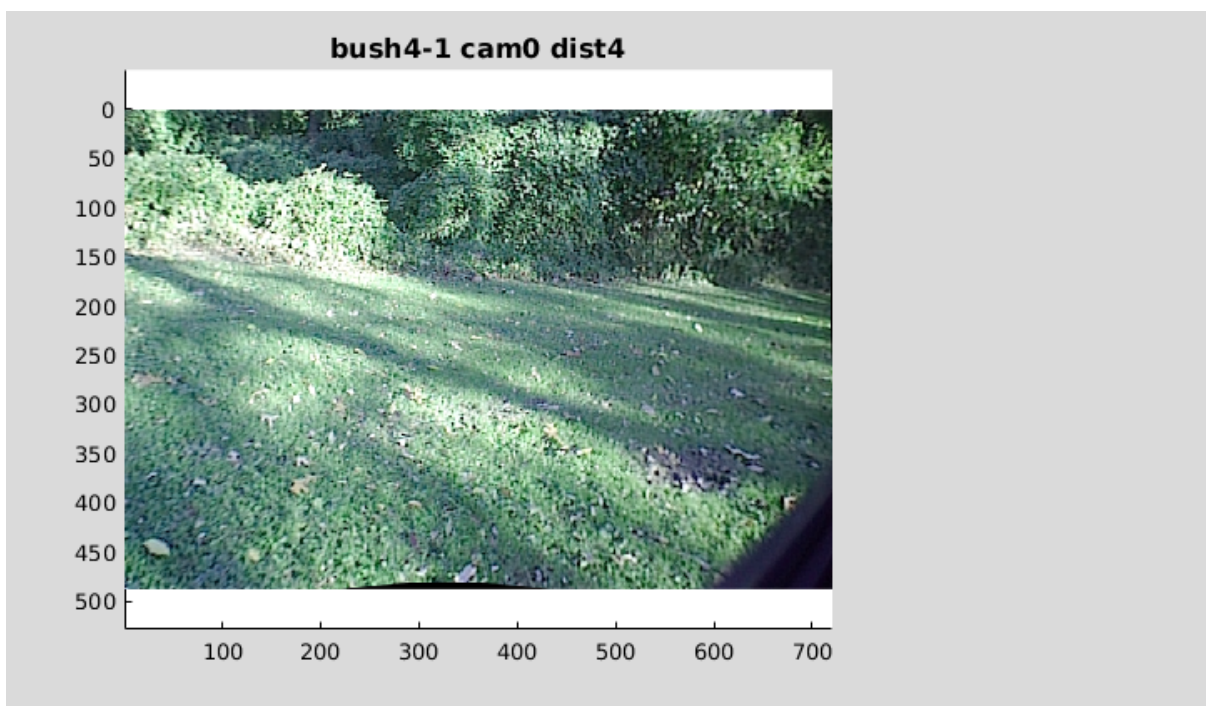


(a)

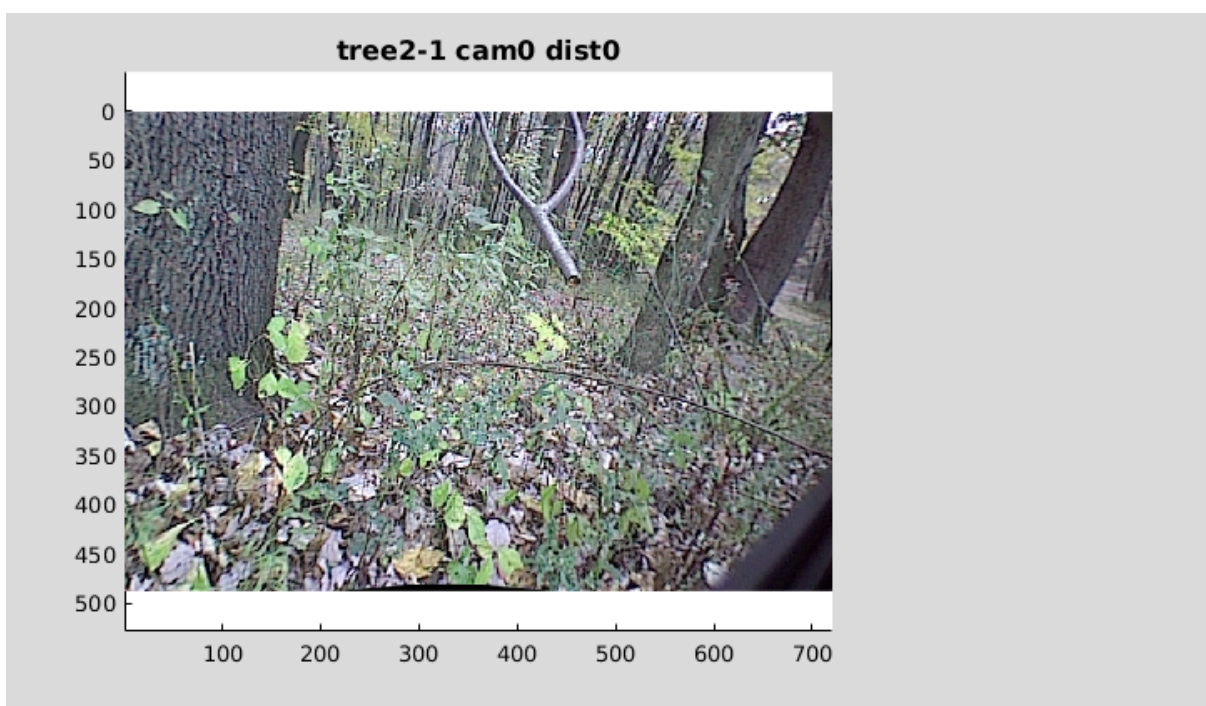


(b)

Figure 9.19: Pixim Images for Previous Locales (sea2-11, wood1-7, wood1-8, bush4-1, tree2-1), Continued, Images 8 - 9

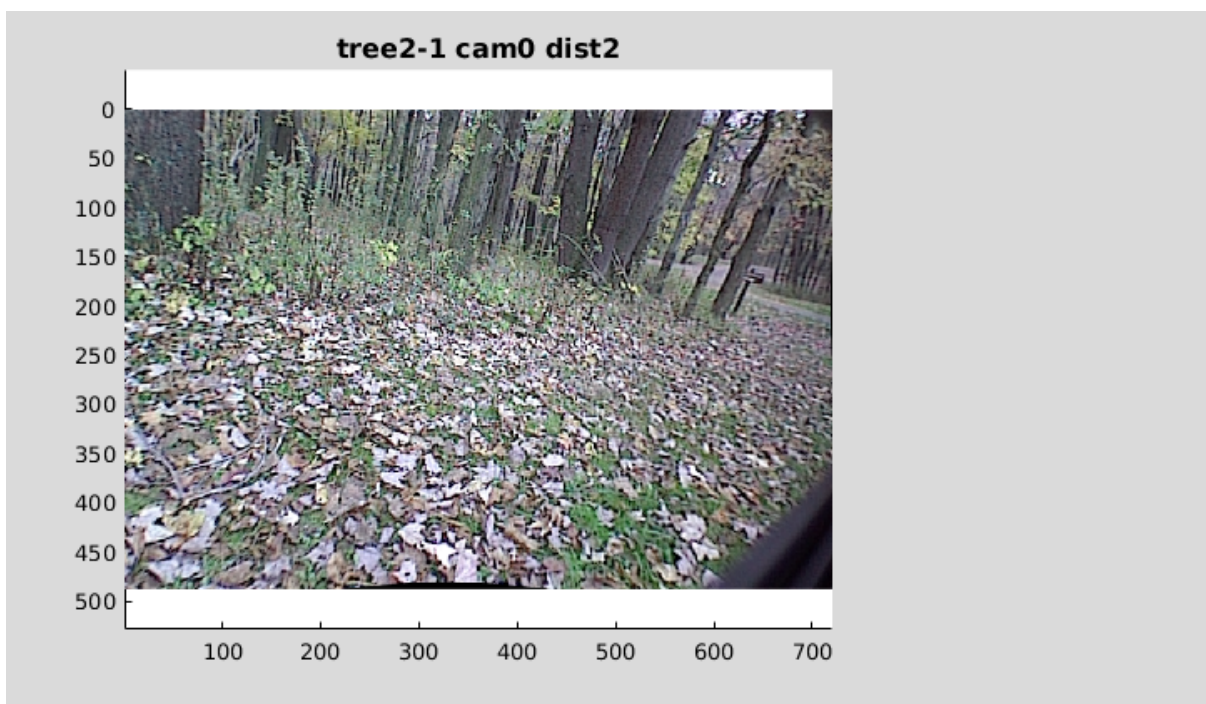


(a)

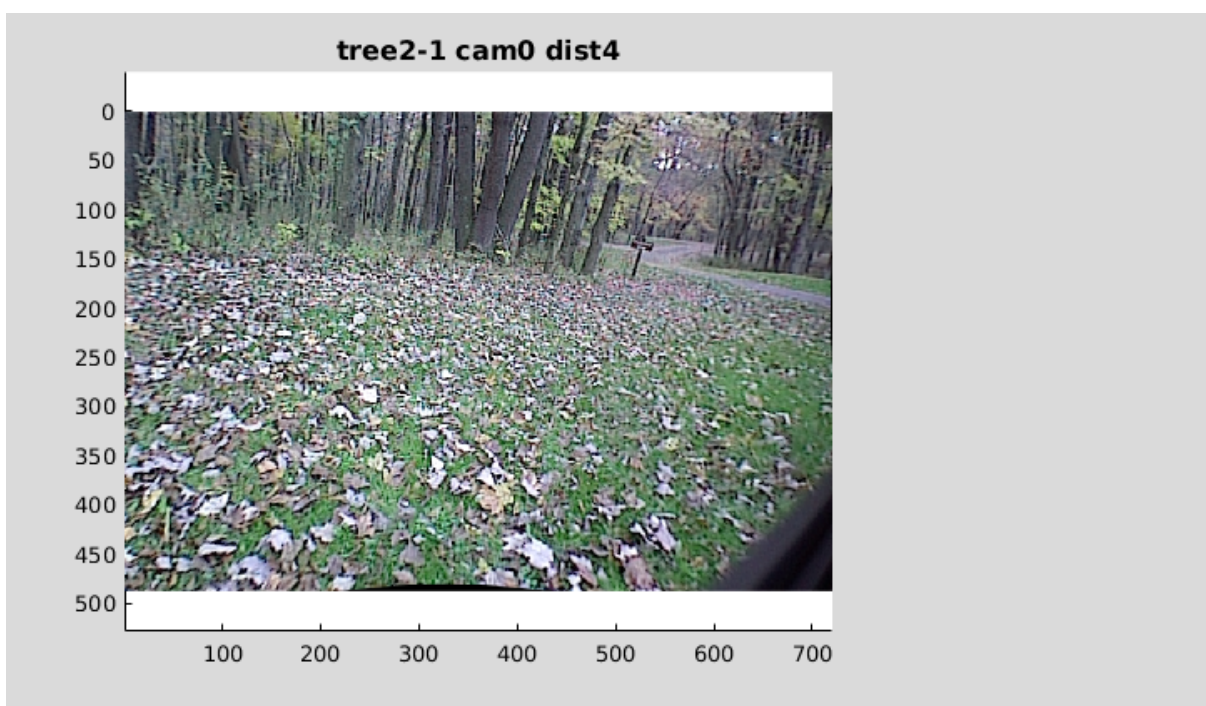


(b)

Figure 9.20: Pixim Images for Previous Locales (sea2-11, wood1-7, wood1-8, bush4-1, tree2-1), Continued, Images 10 - 11



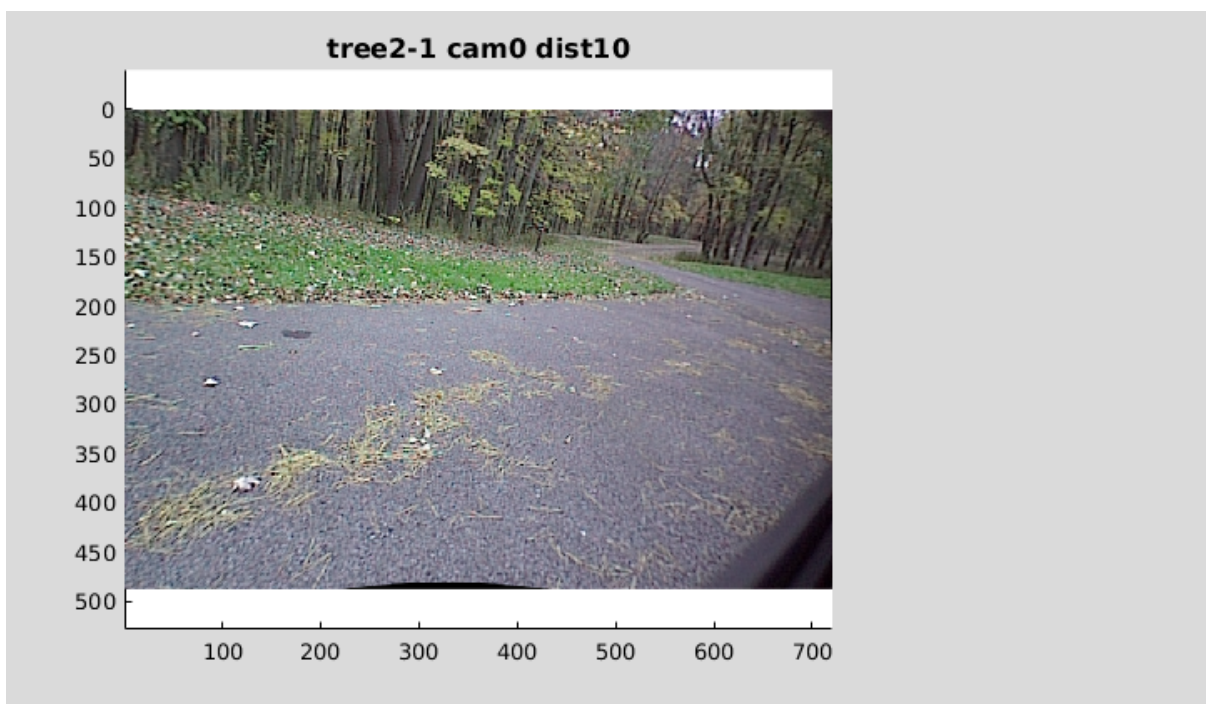
(a)



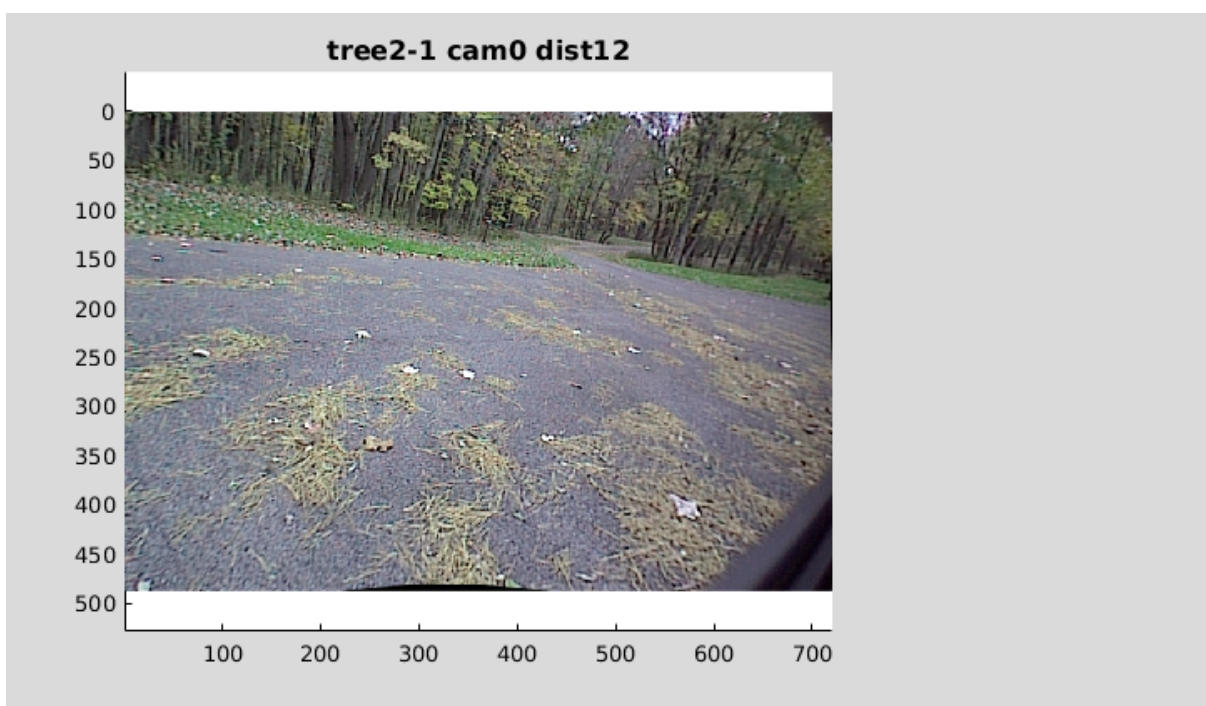
(b)

Figure 9.21: Pixim Images for Previous Locales (sea2-11, wood1-7, wood1-8, bush4-1, tree2-1), Continued, Images 12 - 13





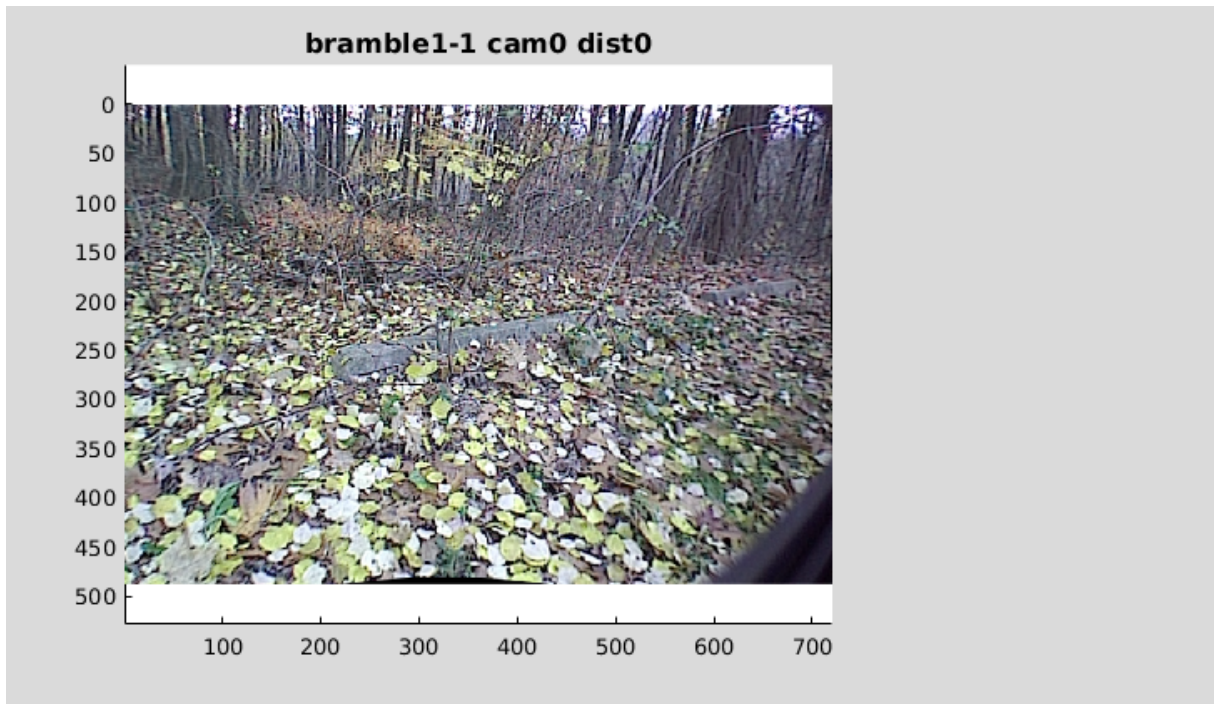
(a)



(b)

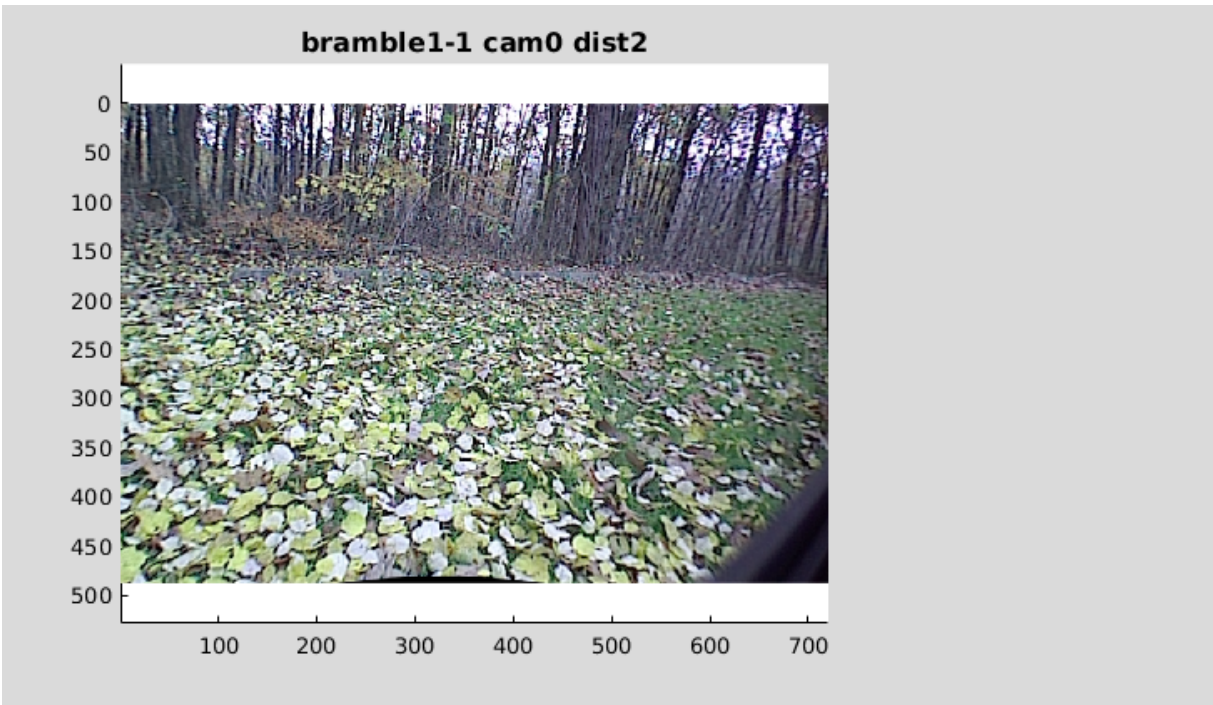
Figure 9.22: Pixim Images for Previous Locales (sea2-11, wood1-7, wood1-8, bush4-1, tree2-1), Continued, Images 14 - 15

## 9.2.2 Pixim Images for Current Locale (bramble1-1)

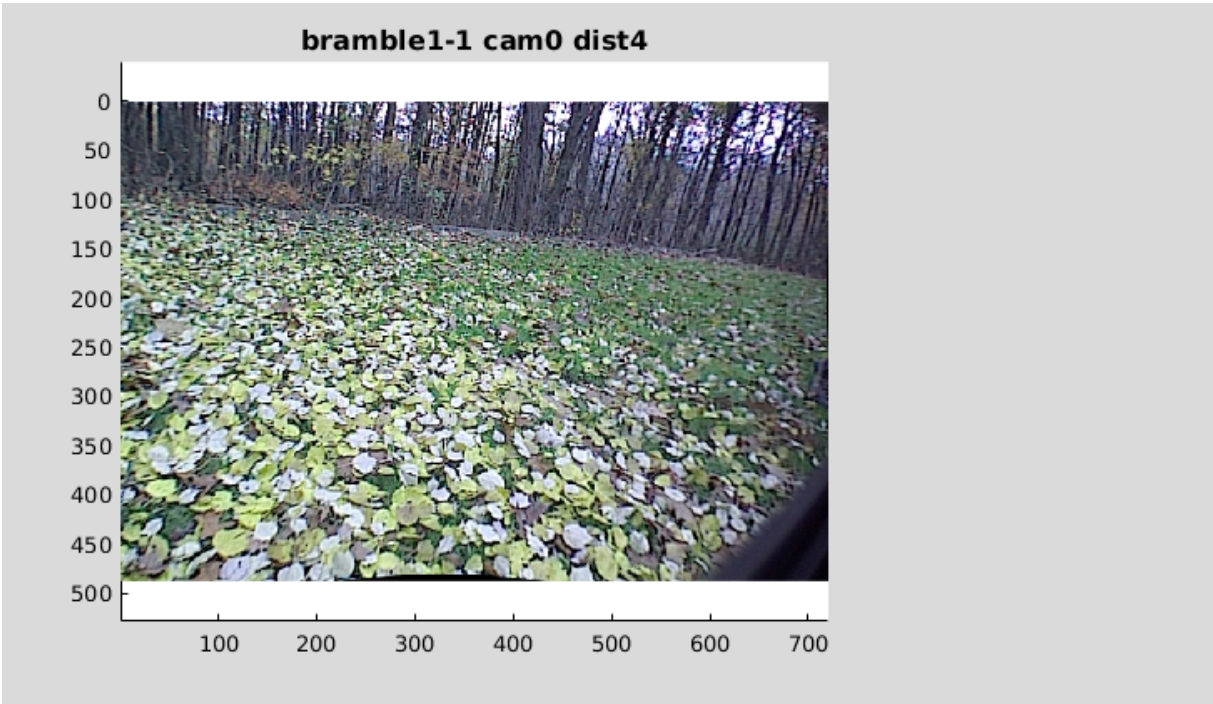


(a)

Figure 9.23: Pixim Images for Current Locale (bramble1-1), Image 1. The current locale is used for testing the vision floor benchmark (refer to images 9.34, 9.35, 9.36 for predictions). This locale is also used for training the self-supervision inside the robot’s path and testing the self-supervision outside the robot’s path. Refer to images 9.37, 9.38, 9.39 for training labels given by the snowball proprioceptive predictions, and then subsequent visual test predictions in 9.40, 9.41, 9.42. Refer to images 9.43, 9.44, 9.45 for training labels given by the (vt500-bumper, adxl-axle-up) proprioceptive predictions, and then subsequent visual test predictions in 9.46, 9.47, 9.48. This locale is also used for training the vision ceiling benchmark inside the robot’s path and testing the vision ceiling benchmark outside the robot’s path. Refer to images 9.49, 9.50, 9.51 for human training labels, and then subsequent visual test predictions in 9.52, 9.53, 9.54.

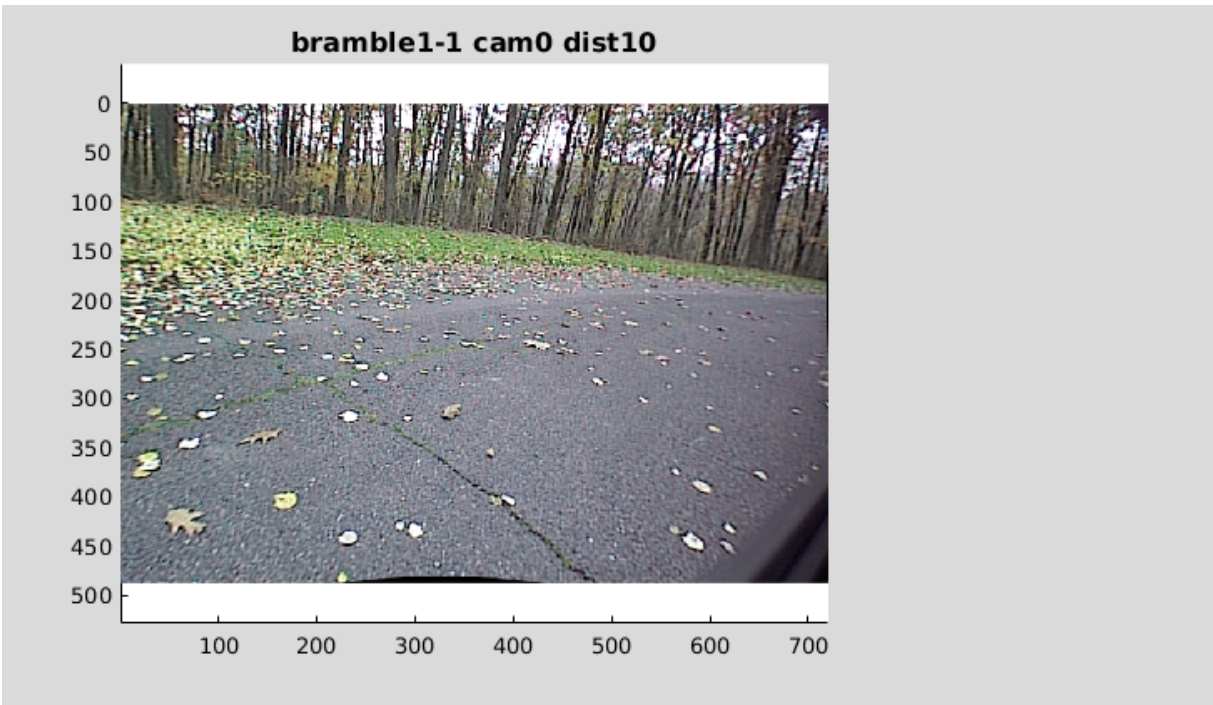


(a)

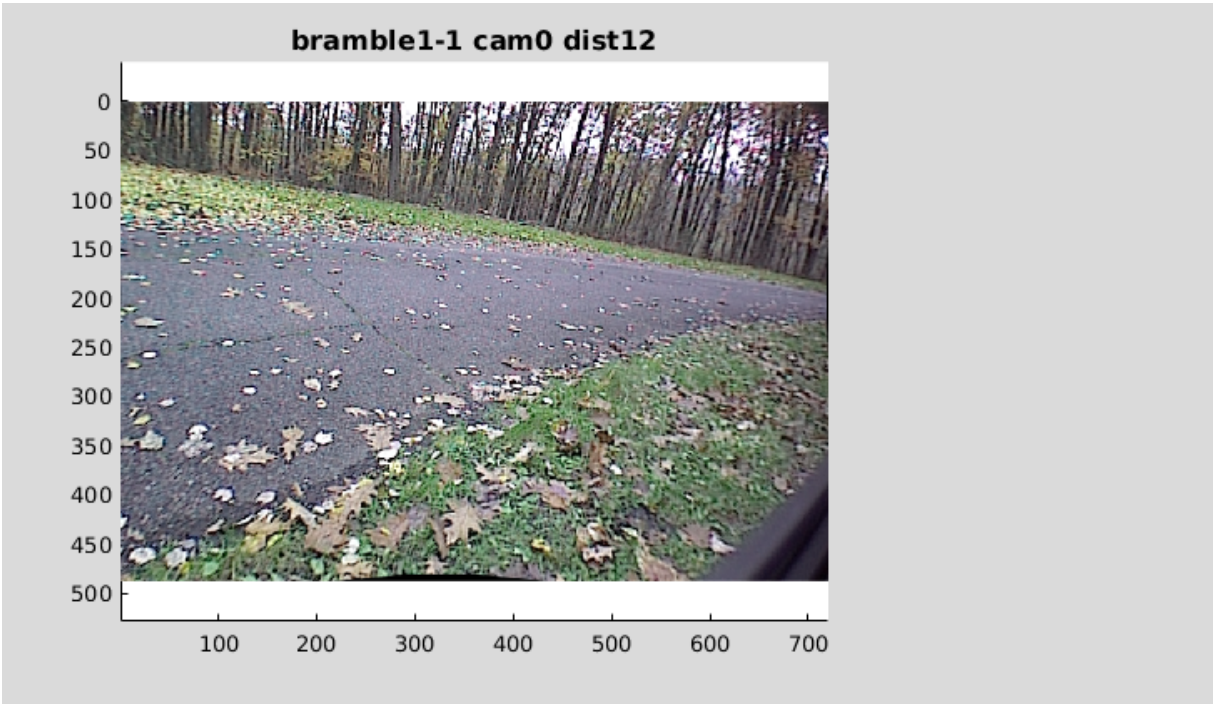


(b)

Figure 9.24: Pixim Images for Current Locale (bramble1-1), Continued, Images 2 - 3



(a)

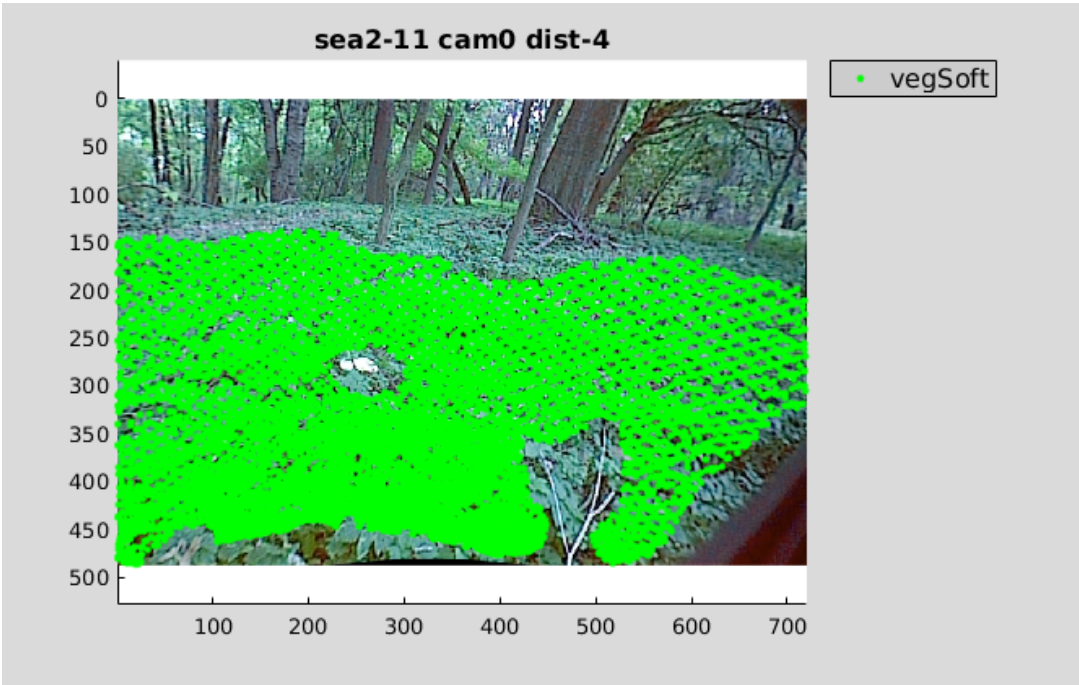


(b)

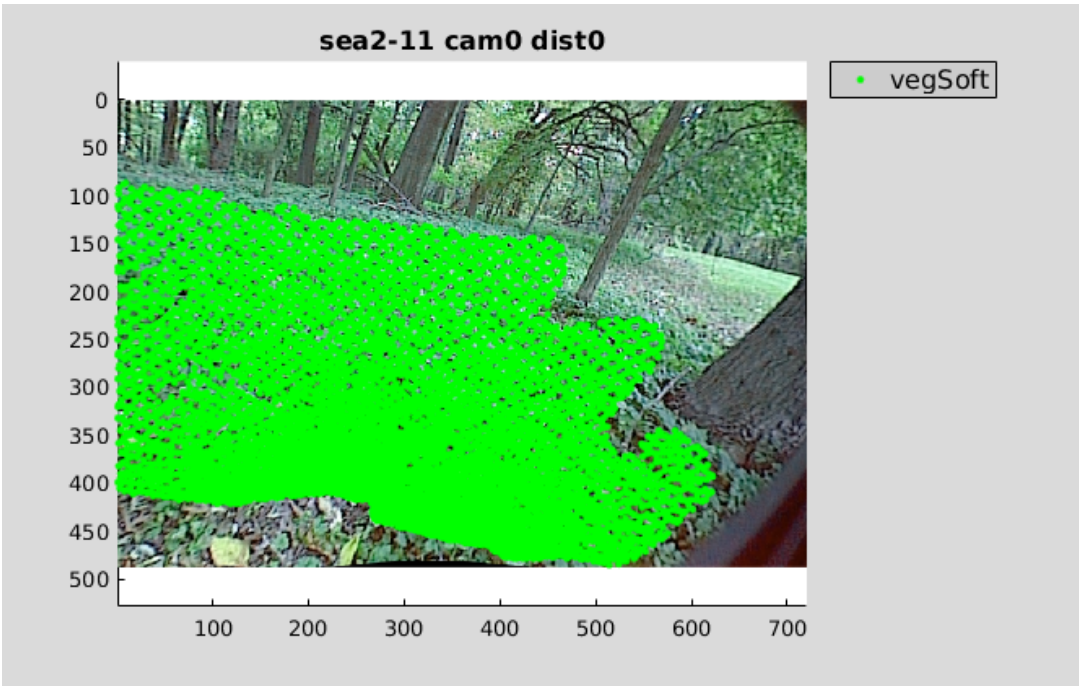
Figure 9.25: Pixim Images for Current Locale (bramble1-1), Continued, Images 4 - 5



### 9.2.3 Vision Floor Benchmark Training Labels

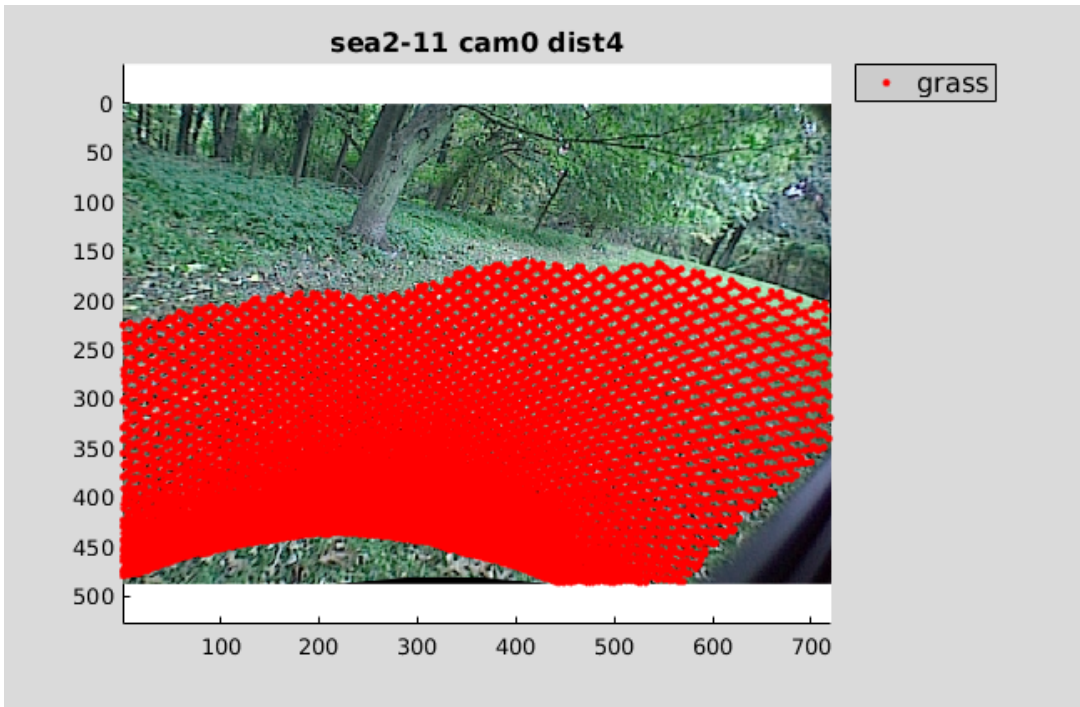


(a)

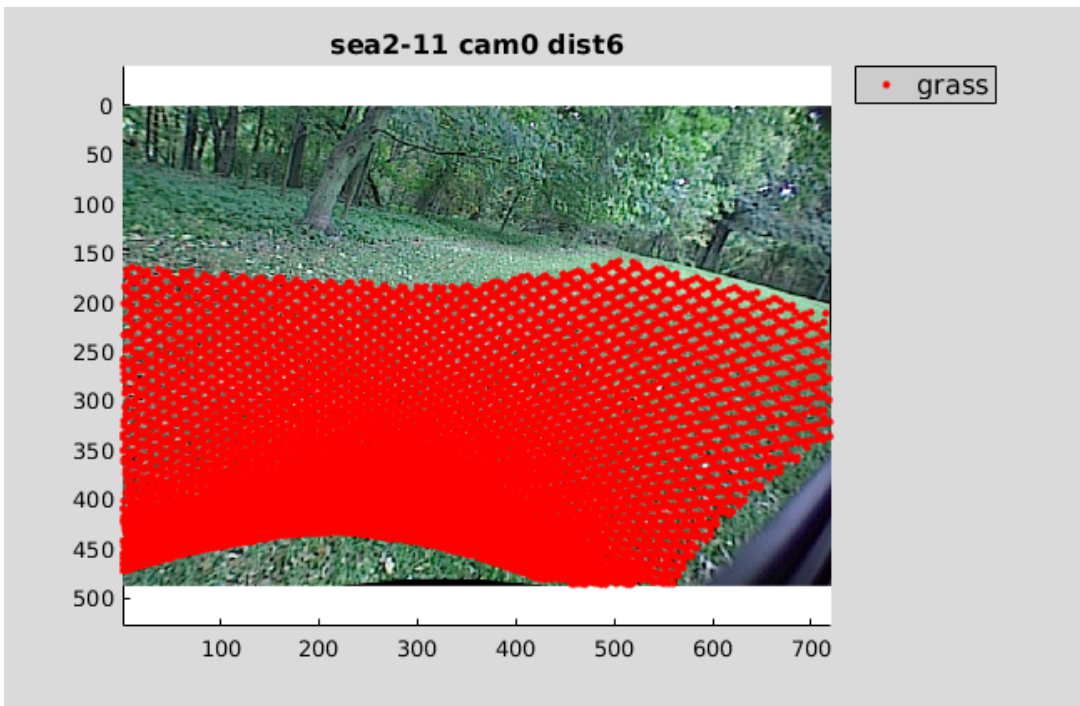


(b)

Figure 9.26: Vision Floor Benchmark Training Labels, Images 1 - 2. Colored points show human labels for supervised training data (locales sea2-11, wood1-7, wood1-8, bush4-1, tree2-1). Refer to 9.15, 9.16, 9.17, 9.18, 9.19, 9.20, 9.21, 9.22 for bare images.

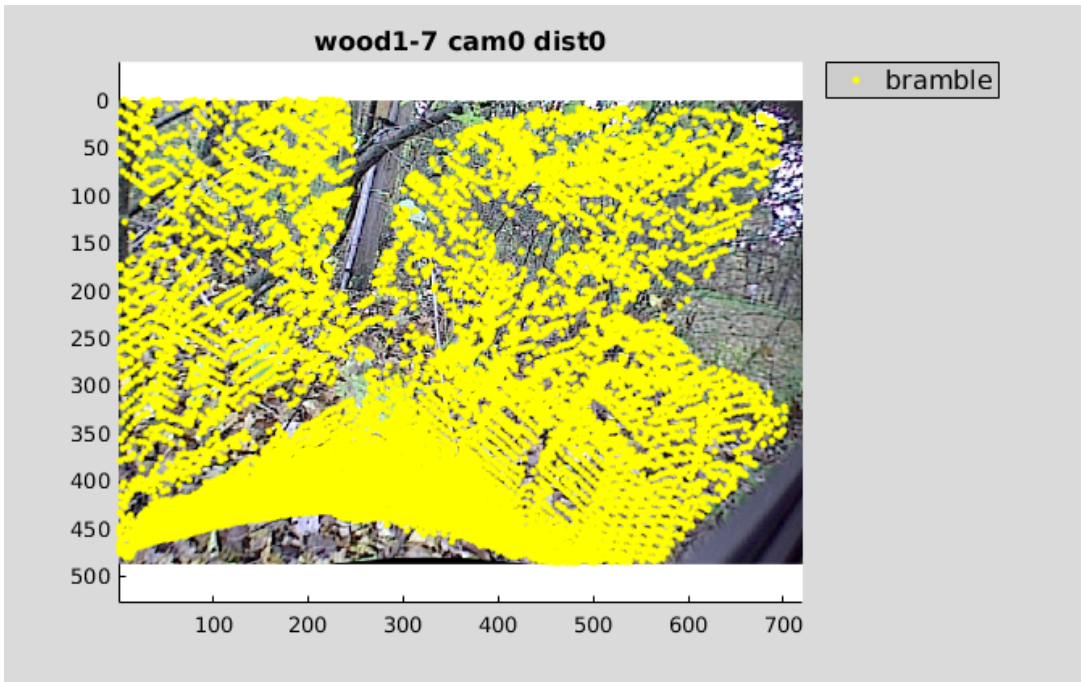


(a)



(b)

Figure 9.27: Vision Floor Benchmark Training Labels, Continued, Images 3 - 4



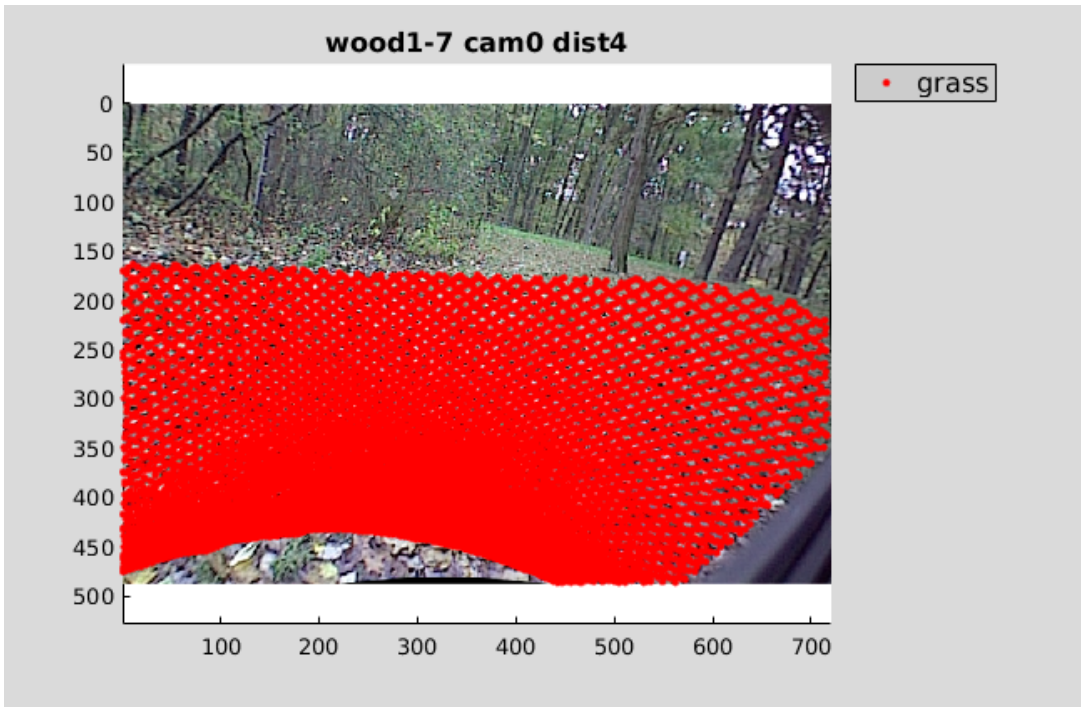
(a)



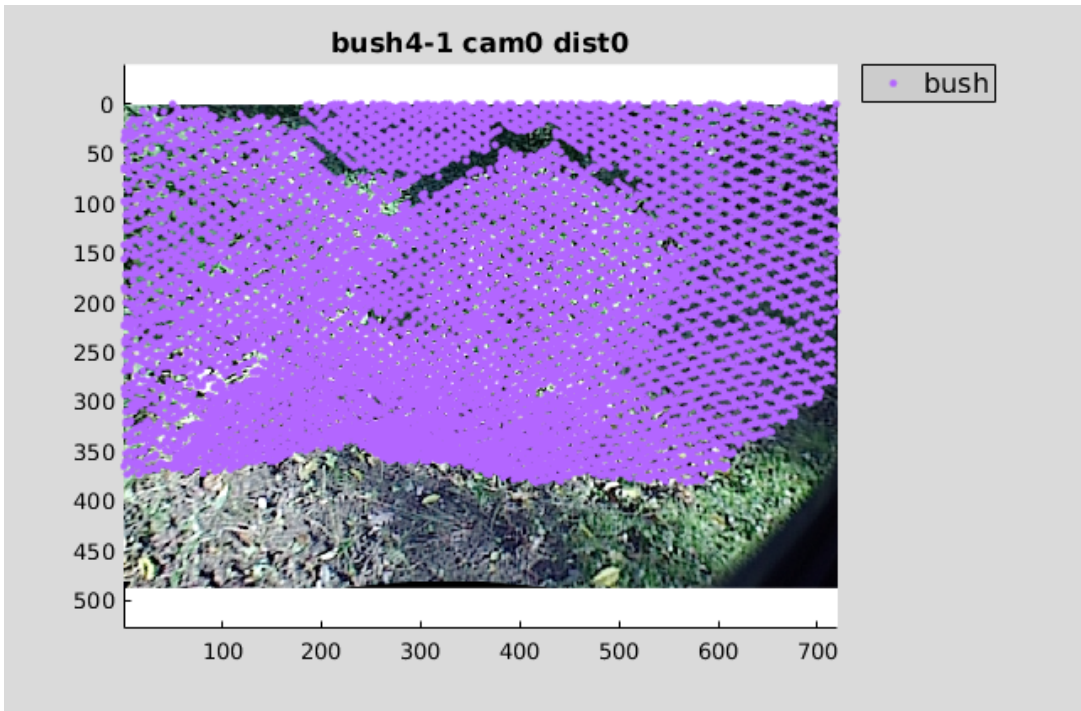
(b)

Figure 9.28: Vision Floor Benchmark Training Labels, Continued, Images 5 - 6



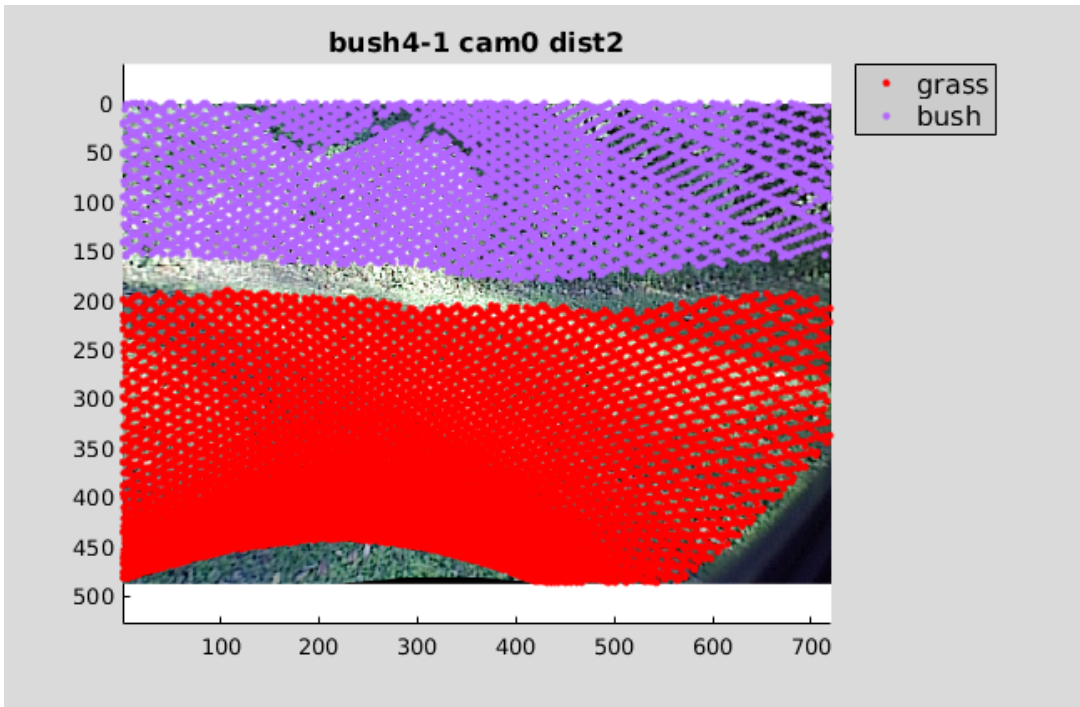


(a)

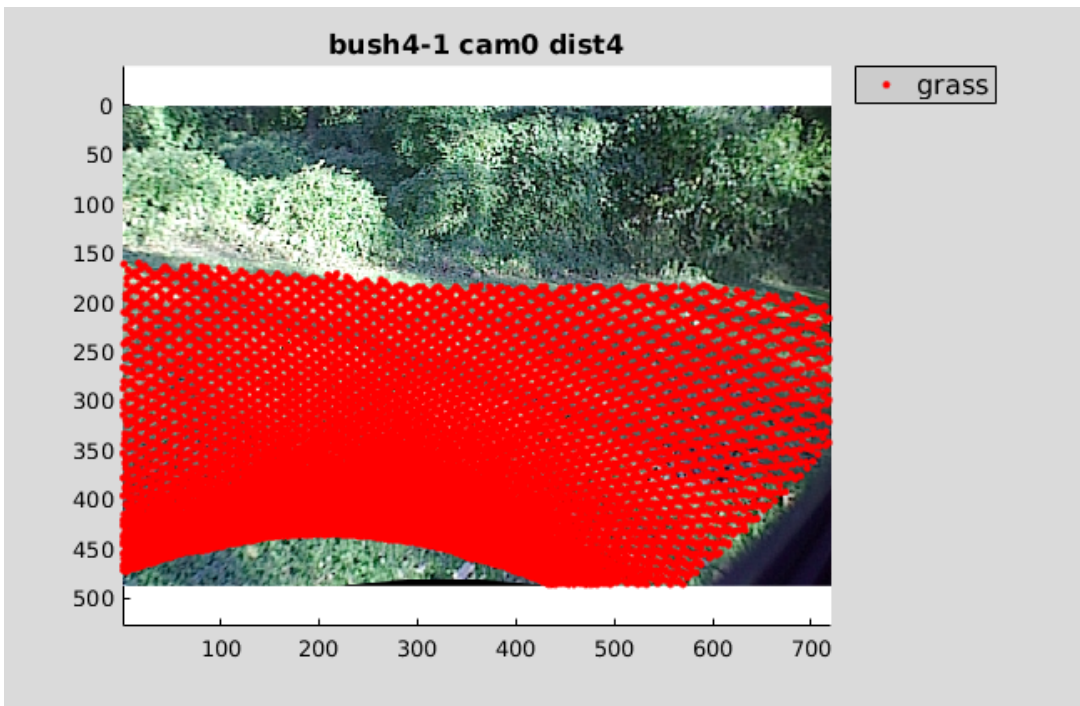


(b)

Figure 9.29: Vision Floor Benchmark Training Labels, Continued, Images 7 - 8



(a)

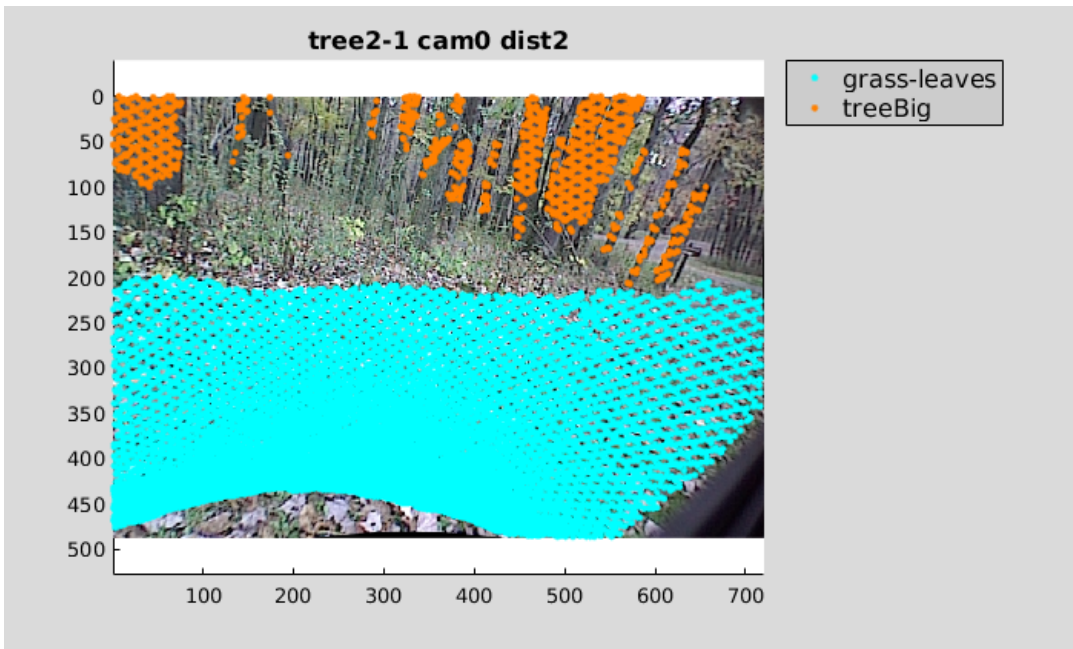


(b)

Figure 9.30: Vision Floor Benchmark Training Labels, Continued, Images 9 - 10



(a)



(b)

Figure 9.31: Vision Floor Benchmark Training Labels, Continued, Images 11 - 12

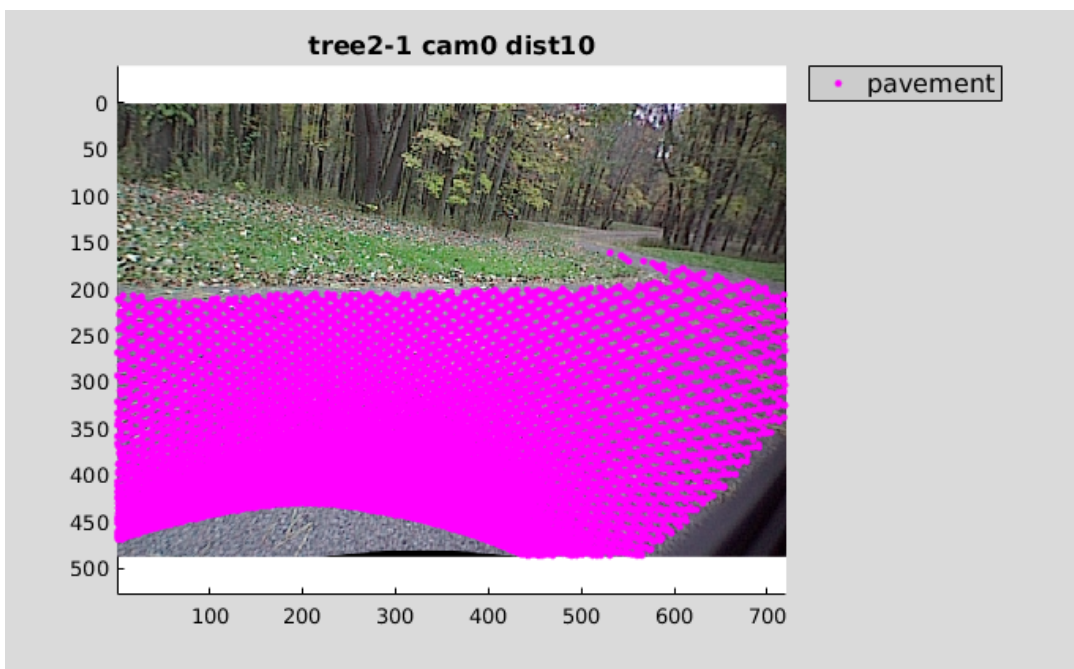
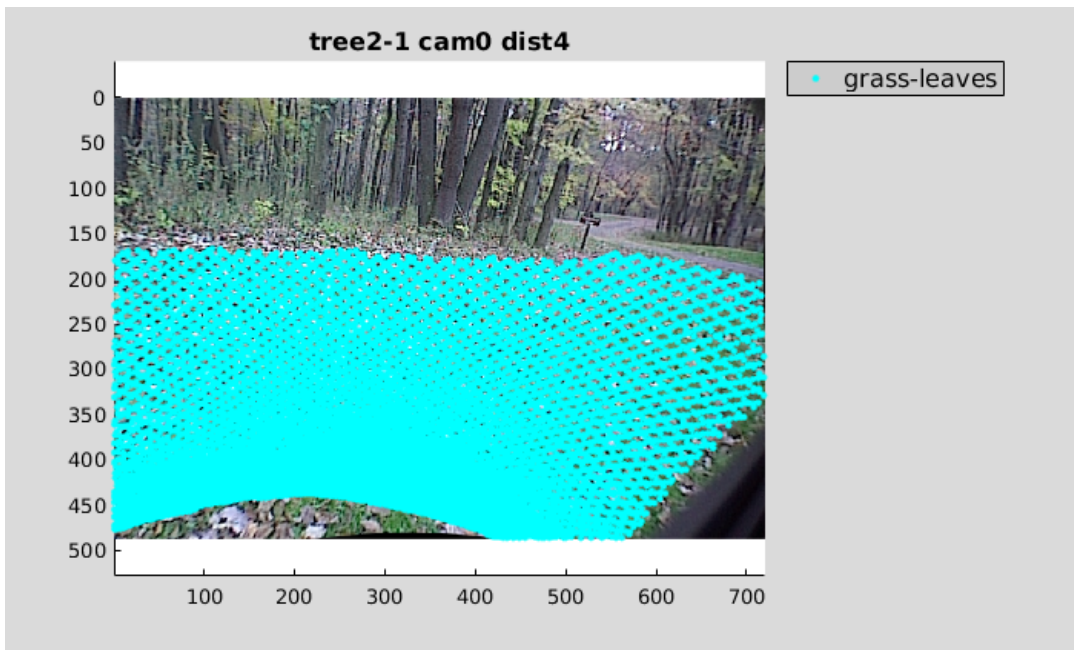
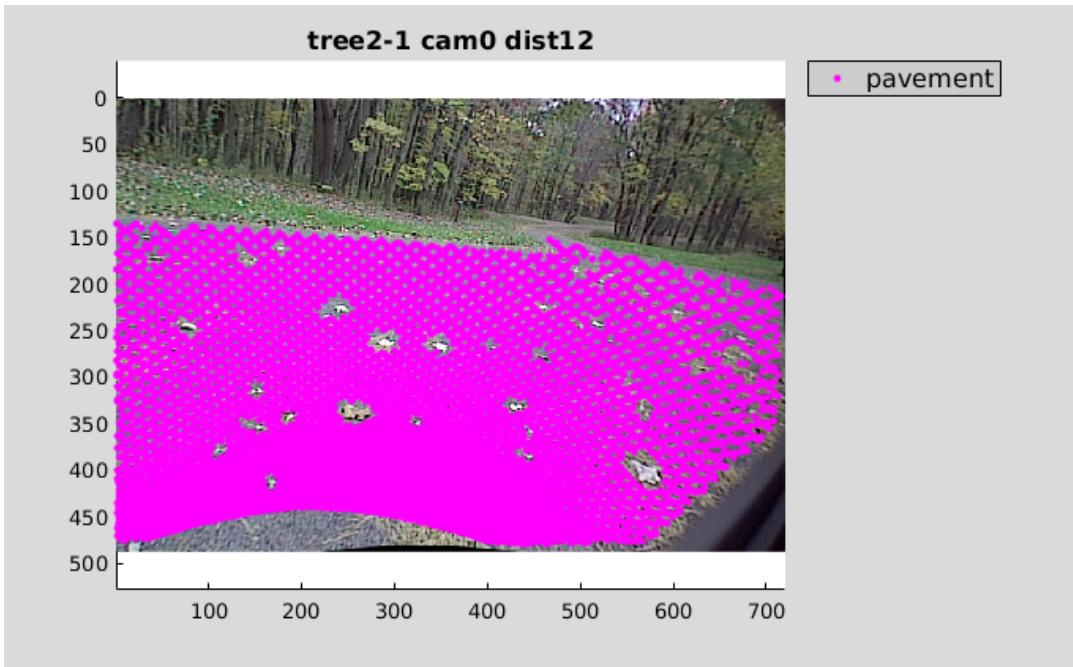


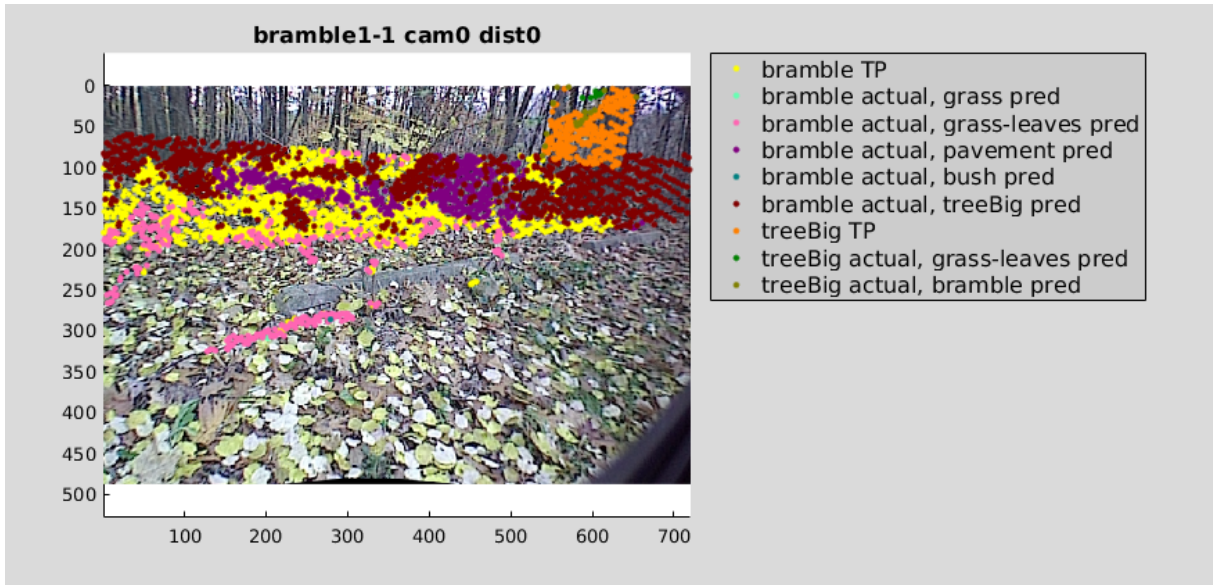
Figure 9.32: Vision Floor Benchmark Training Labels, Continued, Images 13 - 14



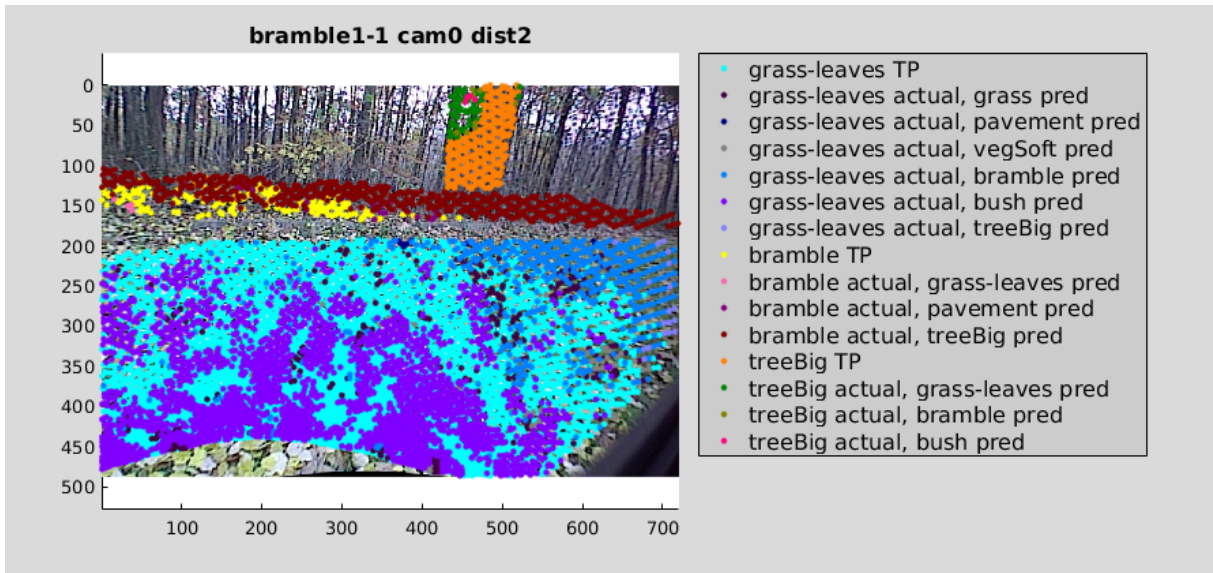
(a)

Figure 9.33: Vision Floor Benchmark Training Labels, Continued, Image 15

## 9.2.4 Vision Floor Benchmark Test Predictions

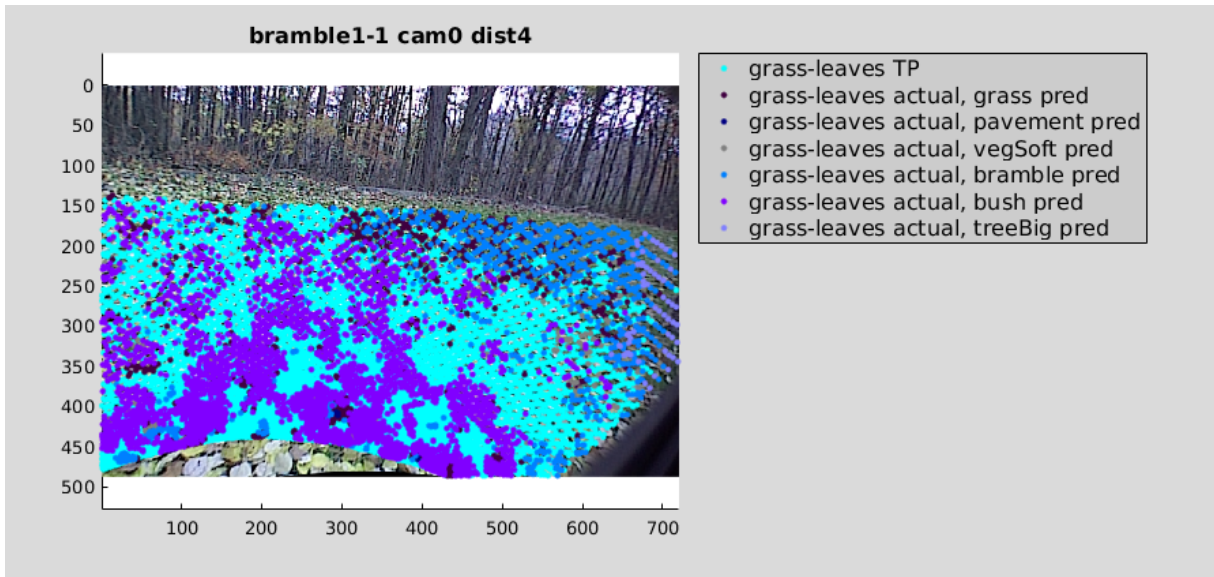


(a)

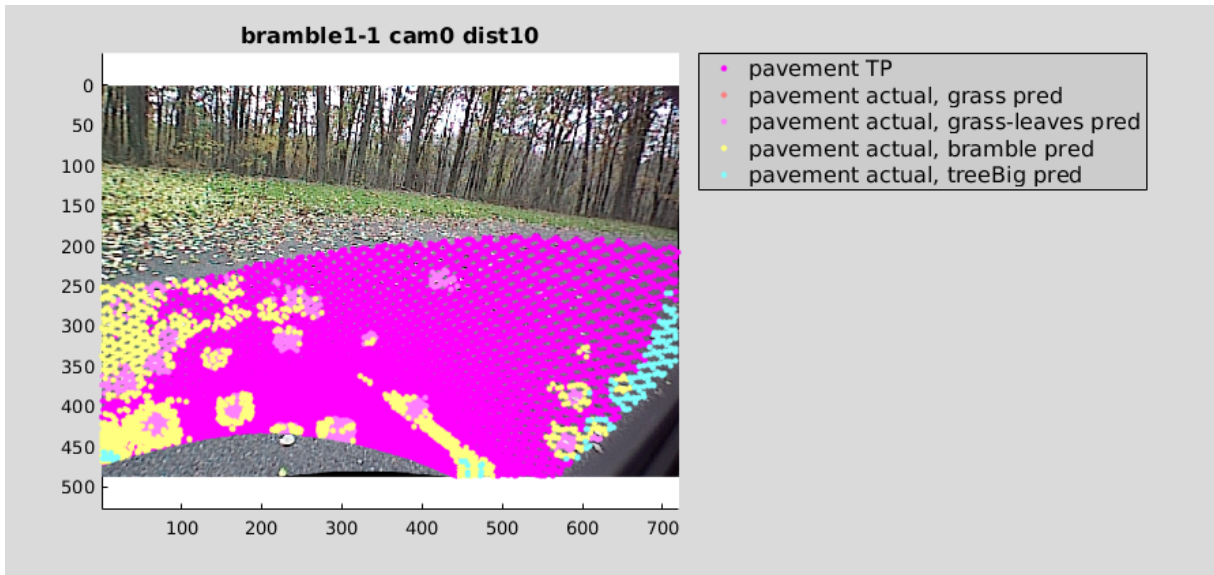


(b)

Figure 9.34: Vision Floor Benchmark Test Predictions, Images 1 - 2. Colored points show visual predictions from supervised classifier on test data (locale bramble1-1). Refer to 9.23, 9.24, 9.25 for corresponding bare test images. The model is trained on the previous set of locales. Refer to 9.15, 9.16, 9.17, 9.18, 9.19, 9.20, 9.21, 9.22 for training images. Refer to images 9.26, 9.27, 9.28, 9.29, 9.30, 9.31, 9.32, 9.33 for training labels.

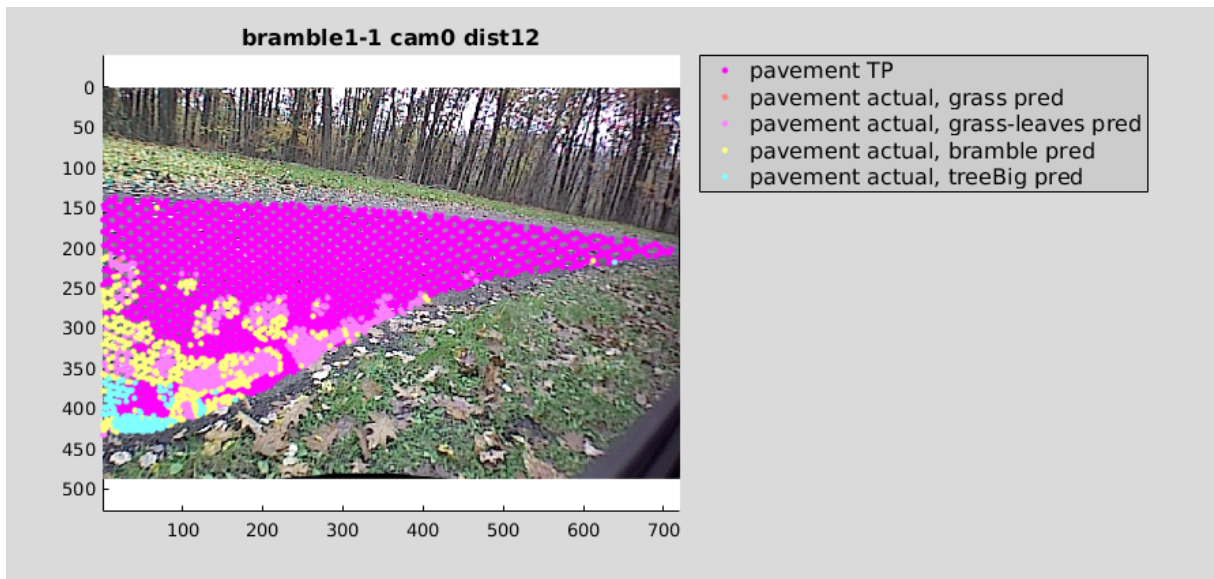


(a)



(b)

Figure 9.35: Vision Floor Benchmark Test Predictions, Continued, Images 3 - 4



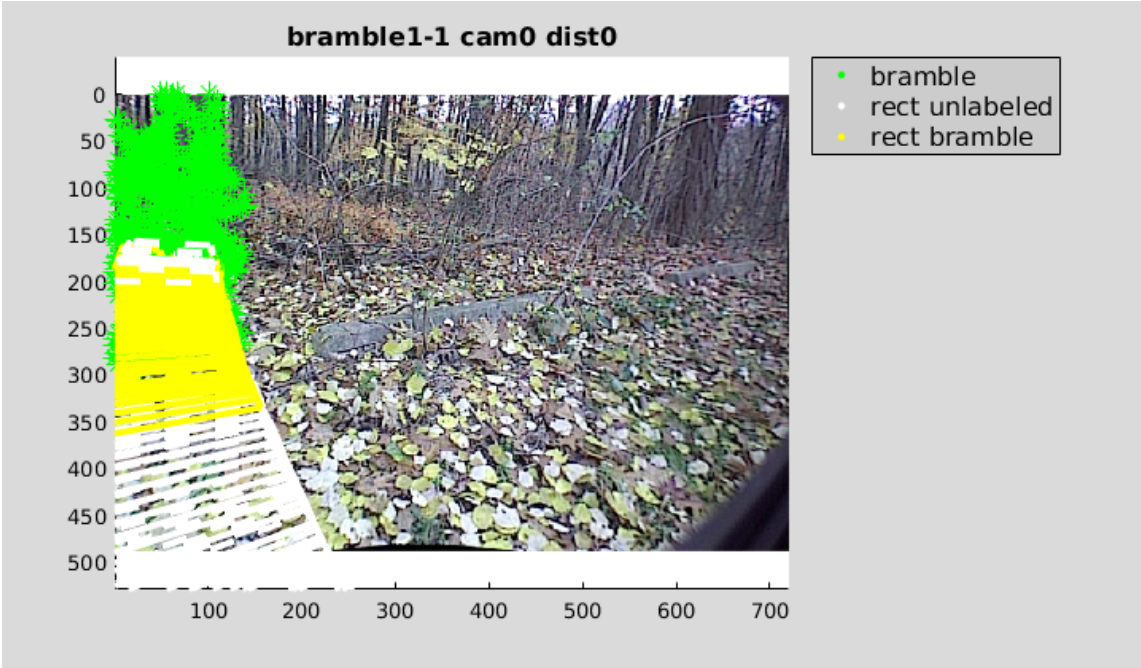
(a)

Figure 9.36: Vision Floor Benchmark Test Predictions, Continued, Image 5

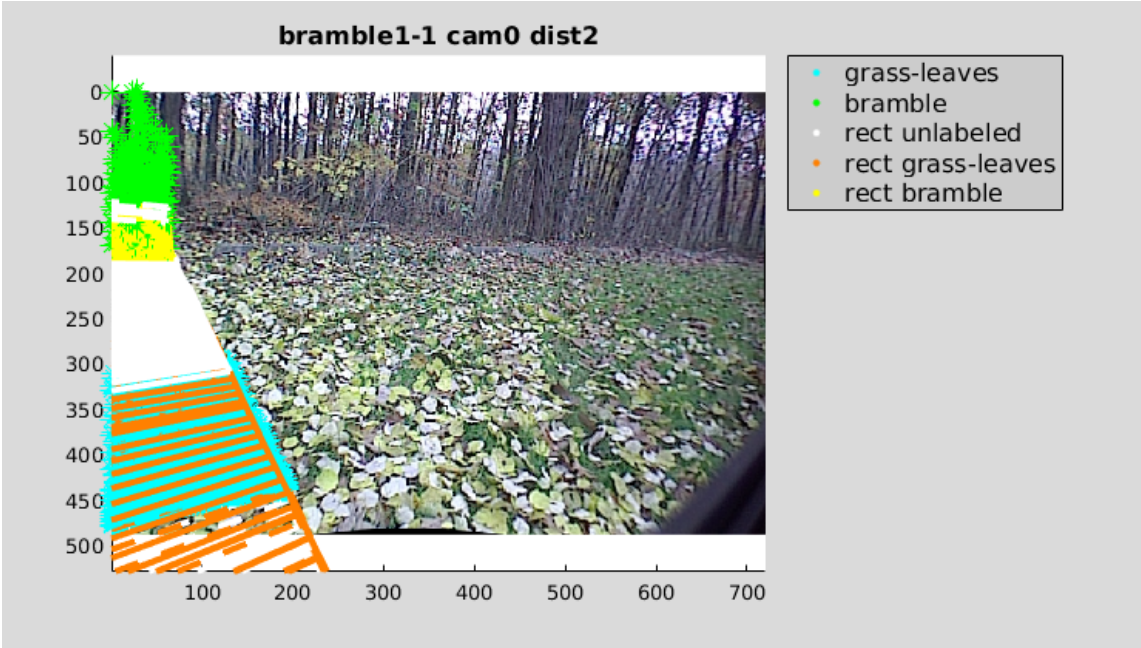




### 9.2.5 Self-Supervision Training Labels (snowball)

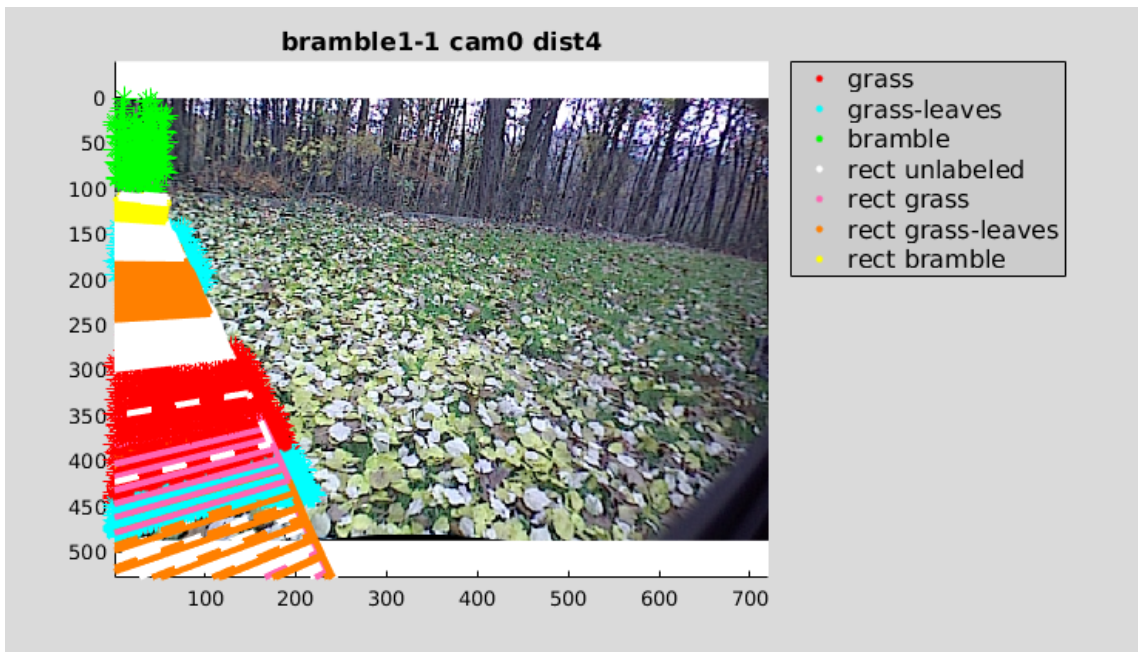


(a)

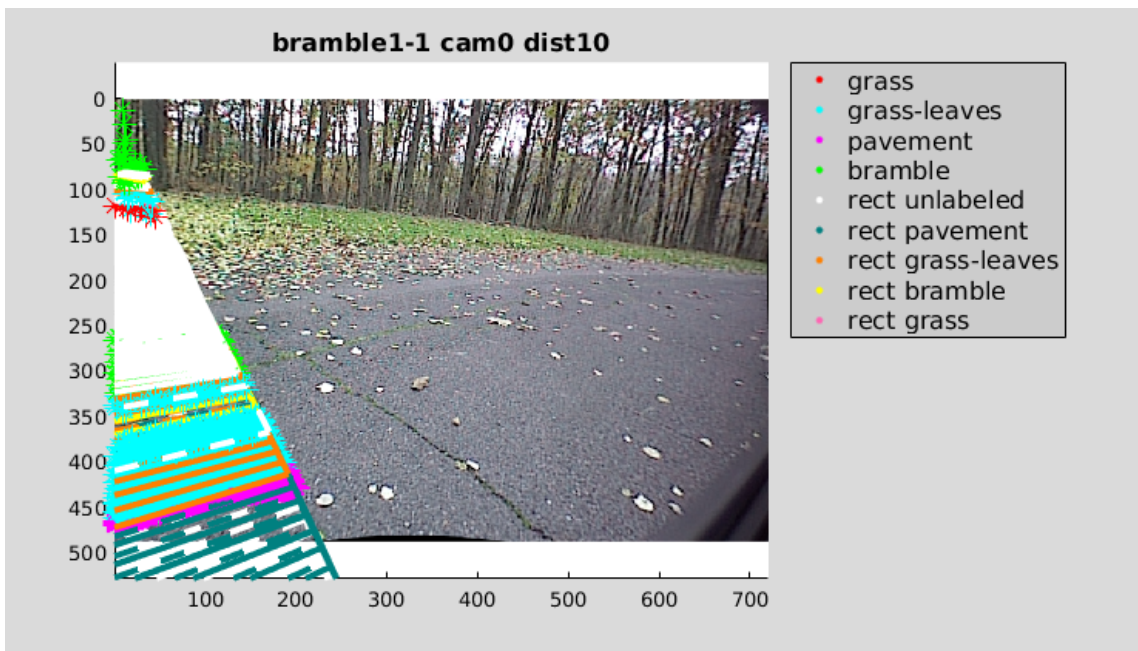


(b)

Figure 9.37: Self-Supervision Training Labels (snowball), Images 1 - 2. Colored points show proprioceptively predicted labels for self-supervised training data (locale bramble1-1, inside the robot’s path). The proprioceptive teacher uses signals from the base snowball sensor. Refer to 9.23, 9.24, 9.25 for bare images. Refer to images 9.40, 9.41, 9.42 for subsequent visual predictions on test data.

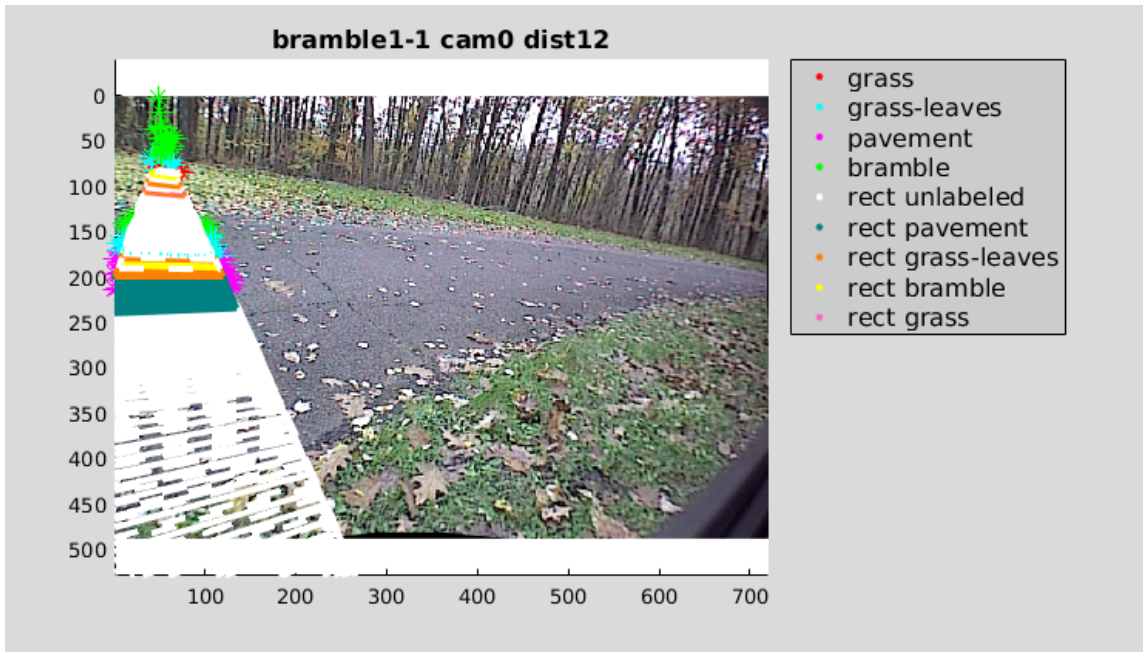


(a)



(b)

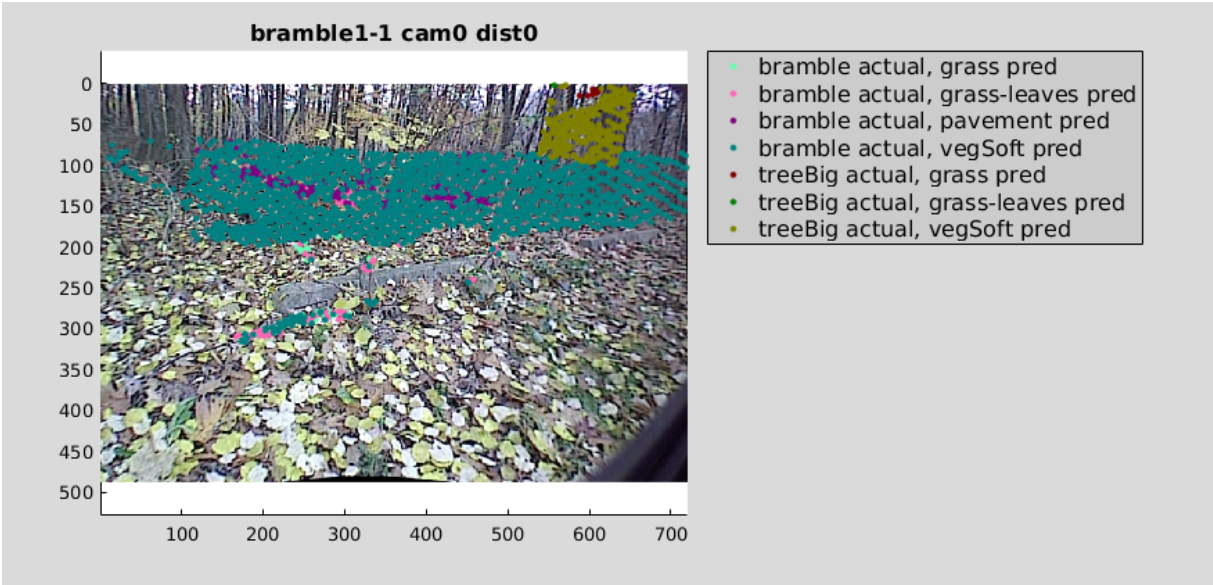
Figure 9.38: Self-Supervision Training Labels (snowball), Continued, Images 3 - 4



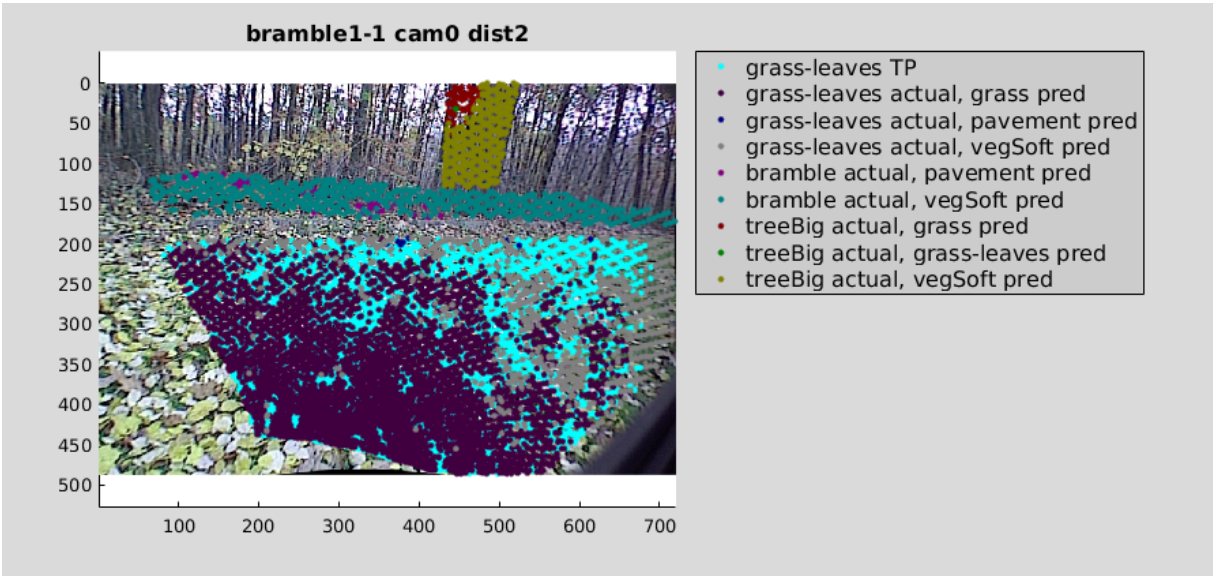
(a)

Figure 9.39: Self-Supervision Training Labels (snowball), Continued, Image 5

### 9.2.6 Self-Supervision Test Predictions (snowball)

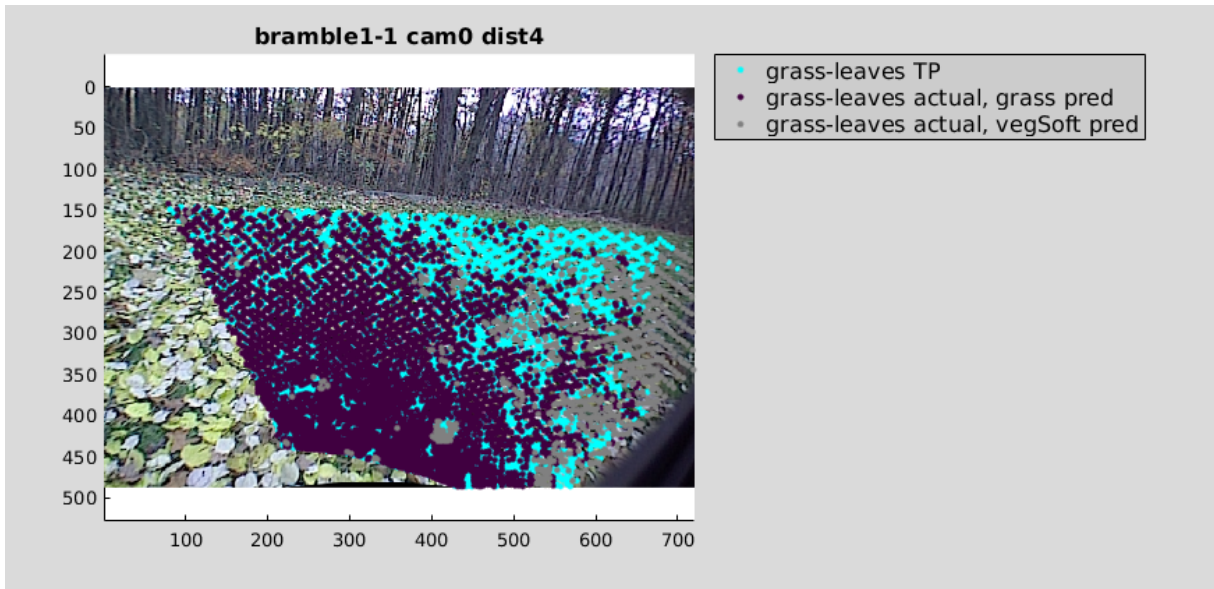


(a)

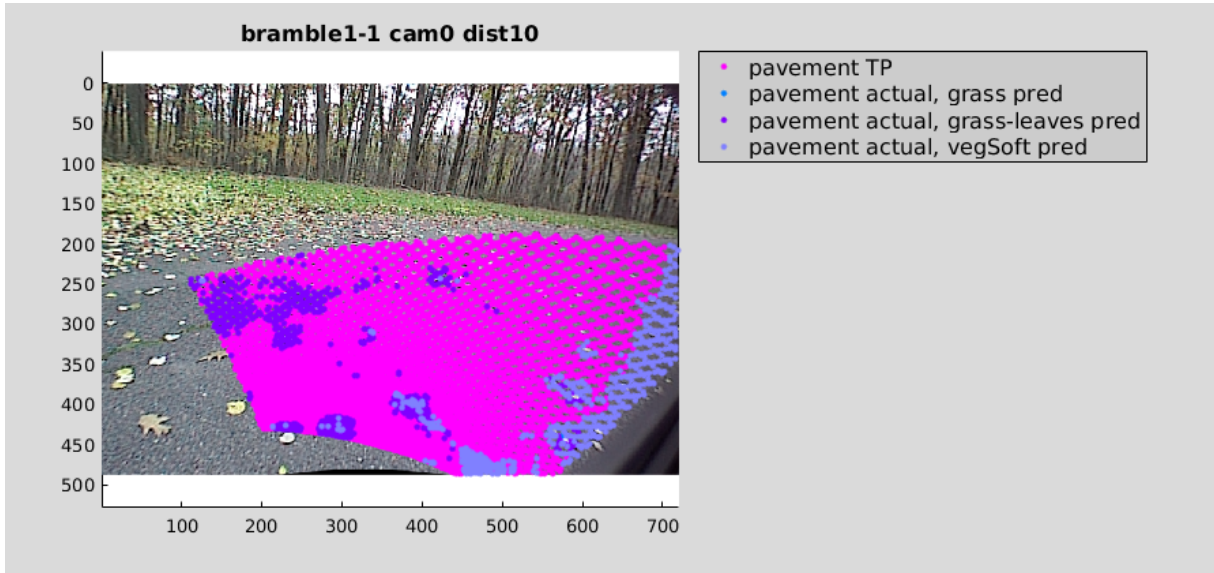


(b)

Figure 9.40: Self-Supervision Test Predictions (snowball), Images 1 - 2. Colored points show visual predictions from self-supervised classifier on test data (locale bramble1-1, outside the robot’s path). The model is trained on data from same locale, inside the robot’s path. The proprioceptive teacher uses signals from the base snowball sensor. Refer to 9.23, 9.24, 9.25 for bare images. Refer to images 9.37, 9.38, 9.39 for self-supervised training labels.

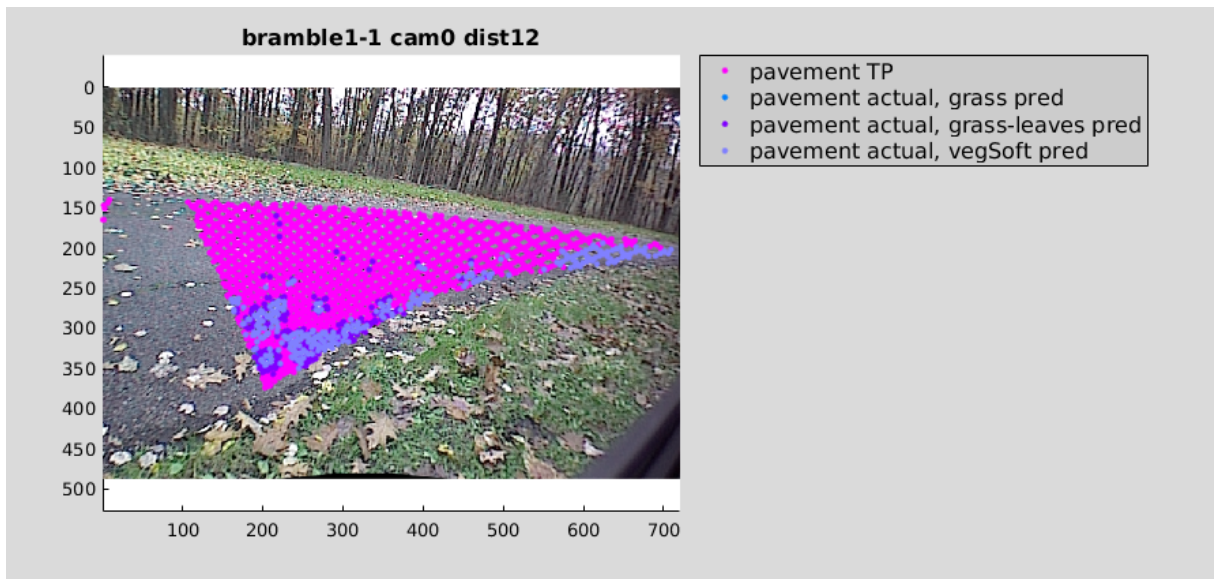


(a)



(b)

Figure 9.41: Self-Supervision Test Predictions (snowball), Continued, Images 3 - 4



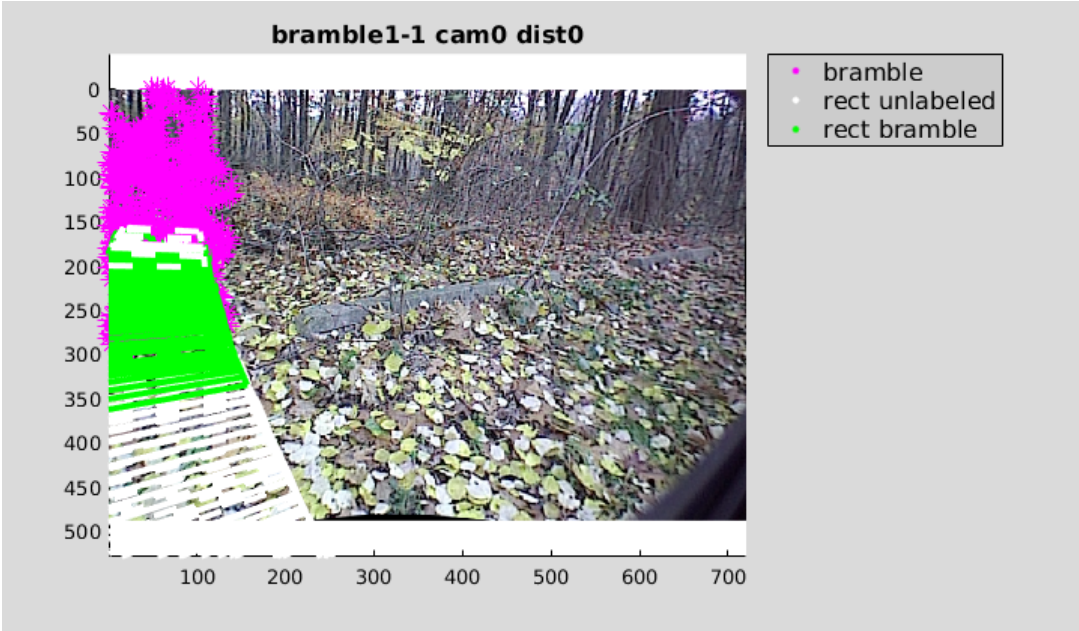
(a)

Figure 9.42: Self-Supervision Test Predictions (snowball), Continued, Image 5

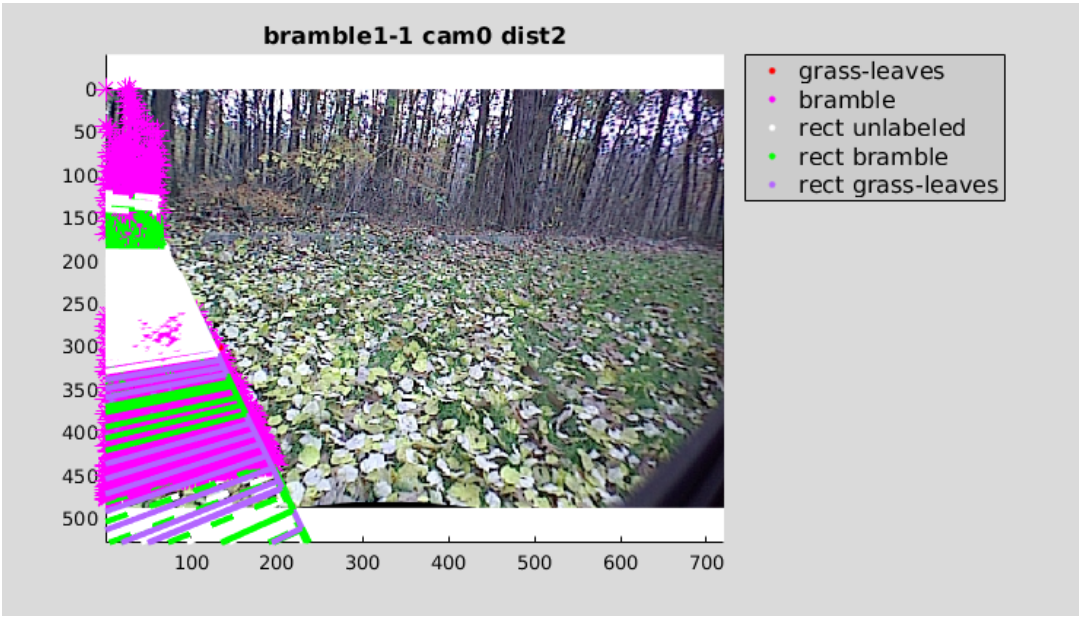




### 9.2.7 Self-Supervision Training Labels (vt500-bumper, adxl-axle-up)

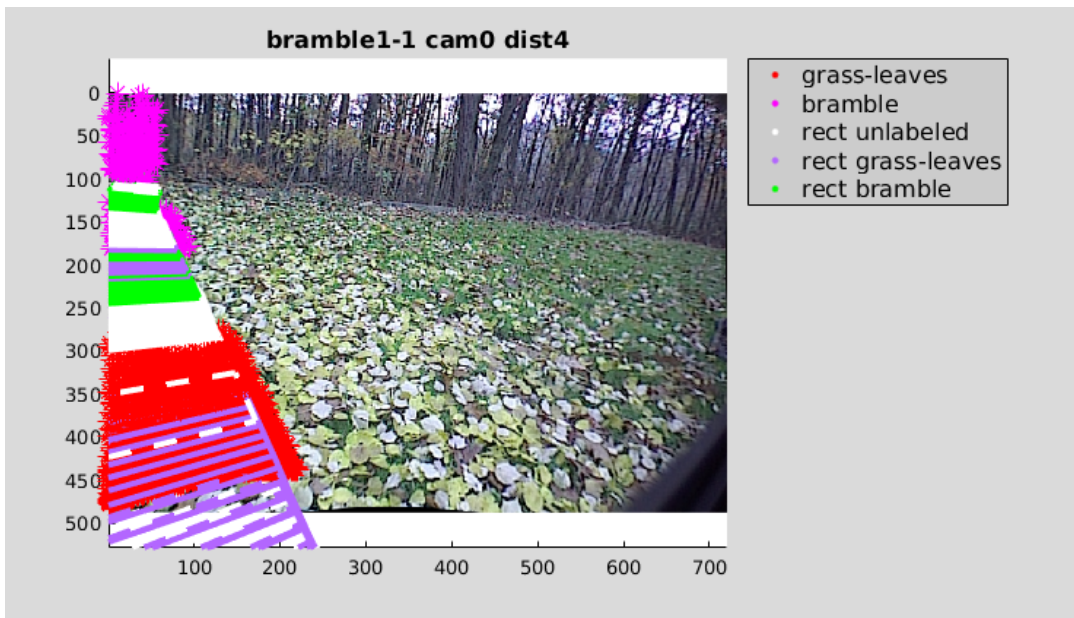


(a)

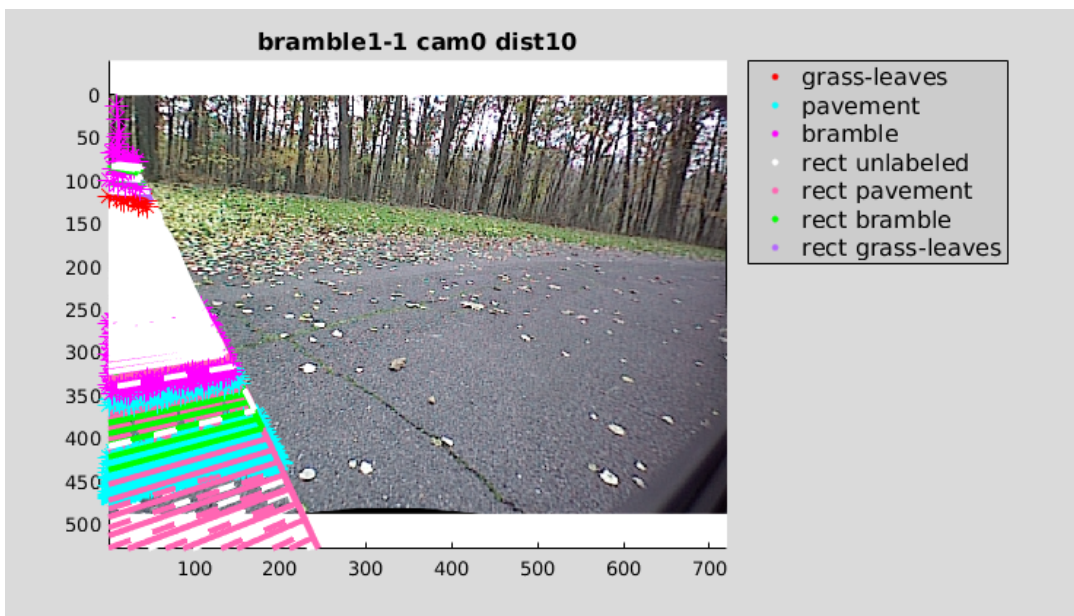


(b)

Figure 9.43: Self-Supervision Training Labels (vt500-bumper, adxl-axle-up), Images 1 - 2. Colored points show proprioceptively predicted labels for self-supervised training data (locale bramble1-1, inside the robot's path). The proprioceptive teacher uses signals from optimal sensor combination (vt500-bumper, adxl-axle-up). Refer to 9.23, 9.24, 9.25 for bare images. Refer to images 9.46, 9.47, 9.48 for subsequent predictions on test data.

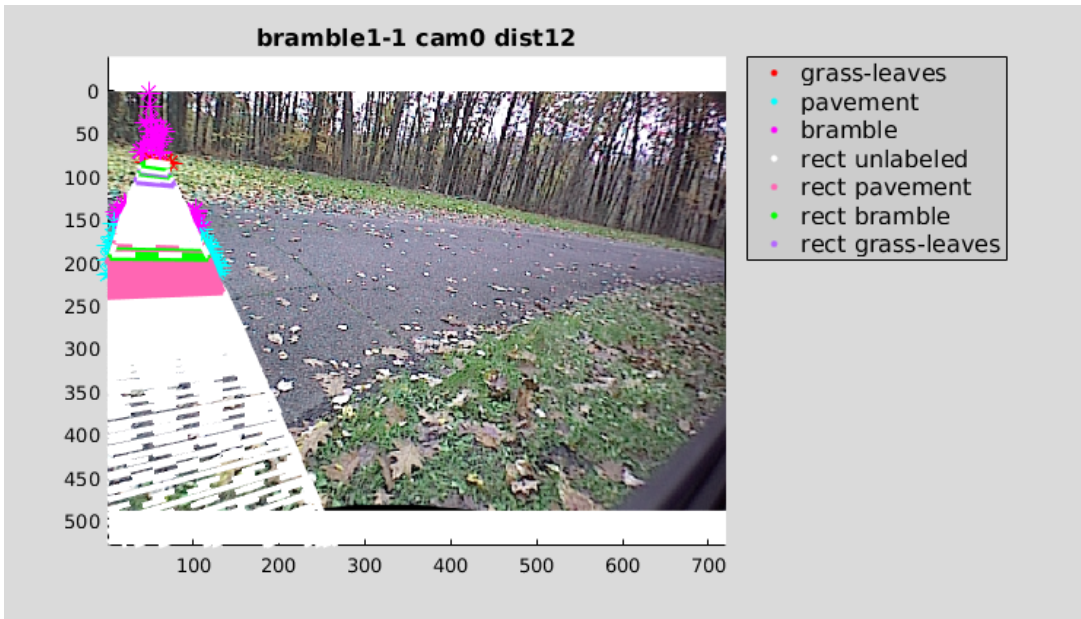


(a)



(b)

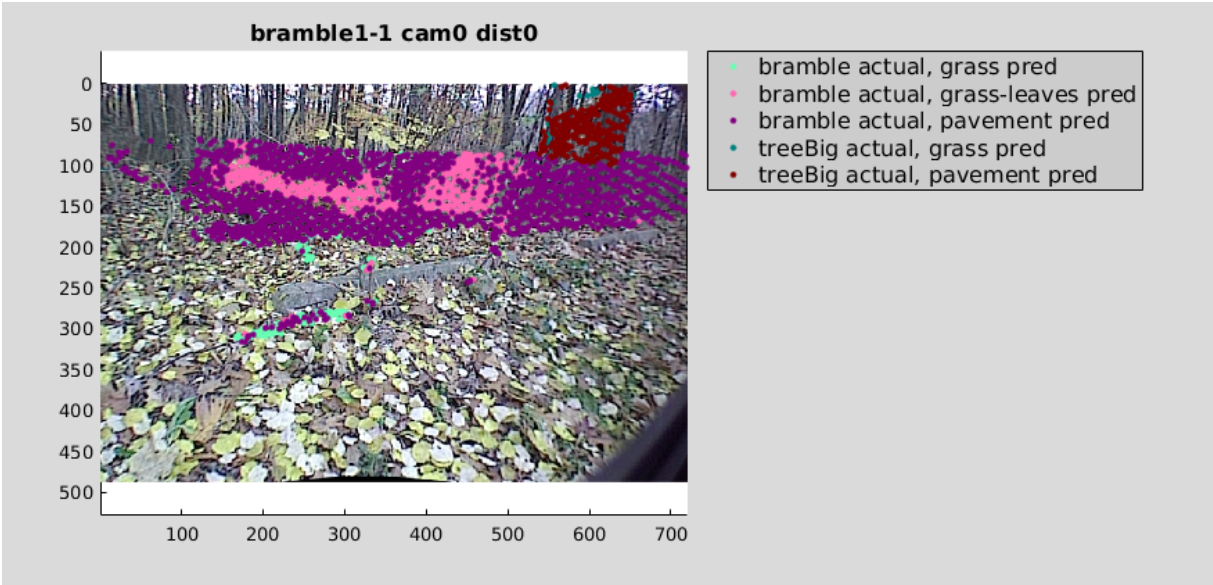
Figure 9.44: Self-Supervision Training Labels (vt500-bumper, adxl-axle-up), Continued, Images 3 - 4



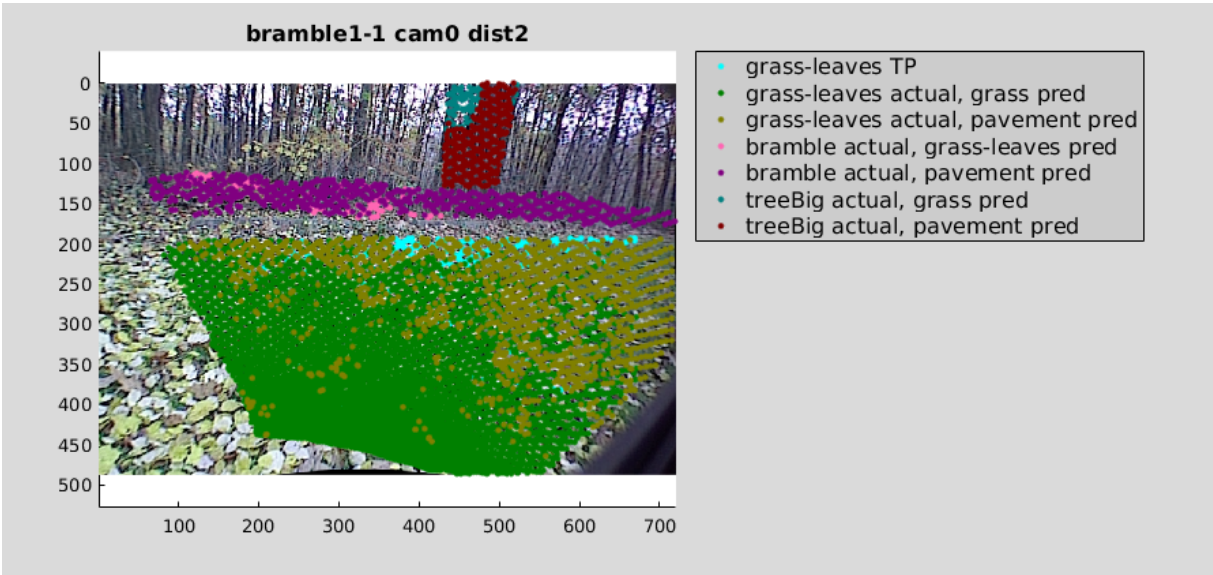
(a)

Figure 9.45: Self-Supervision Training Labels (vt500-bumper, adxl-axle-up), Continued, Image 5

### 9.2.8 Self-Supervision Test Predictions (vt500-bumper, adxl-axle-up)

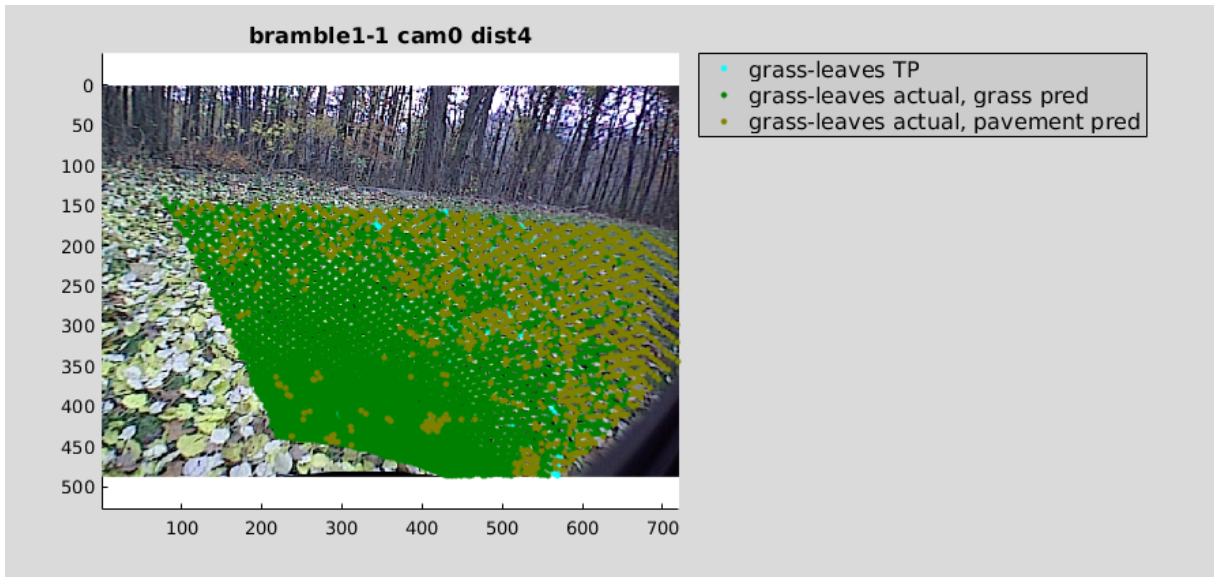


(a)

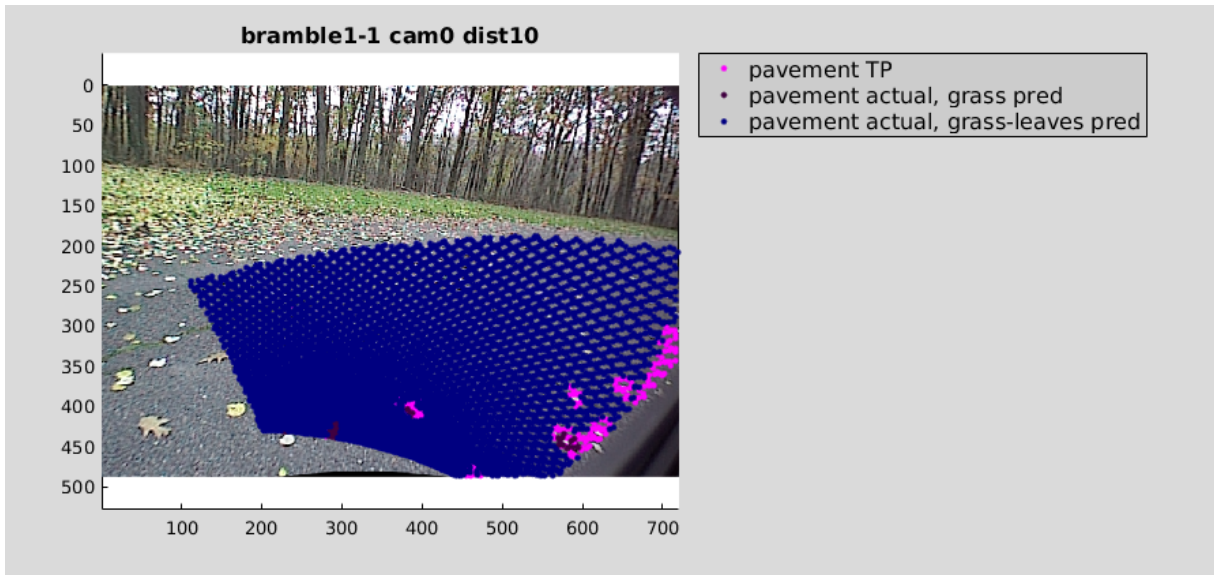


(b)

Figure 9.46: Self-Supervision Test Predictions (vt500-bumper, adxl-axle-up), Images 1 - 2. Colored points show visual predictions from self-supervised classifier on test data (locale bramble1-1, outside the robot’s path). The model is trained on data from same locale, inside the robot’s path. The proprioceptive teacher uses signals from the optimal sensor combination (vt500-bumper, adxl-axle-up). Refer to images 9.23, 9.24, 9.25 for bare images. Refer to 9.43, 9.44, 9.45 for self-supervised training labels.

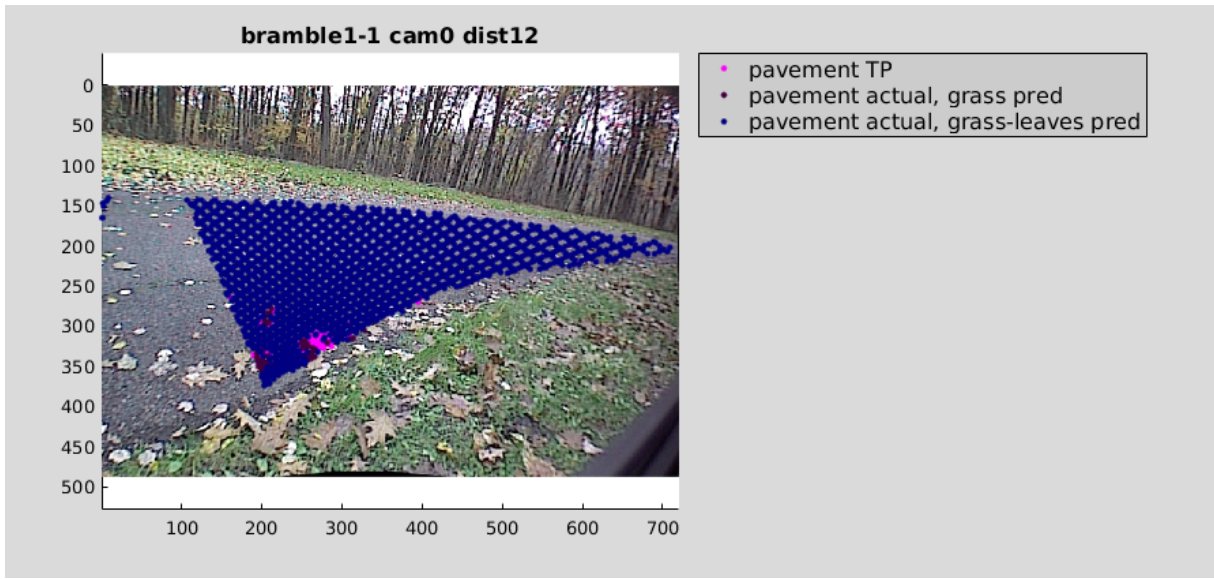


(a)



(b)

Figure 9.47: Self-Supervision Test Predictions (vt500-bumper, adxl-axle-up), Continued, Images 3 - 4

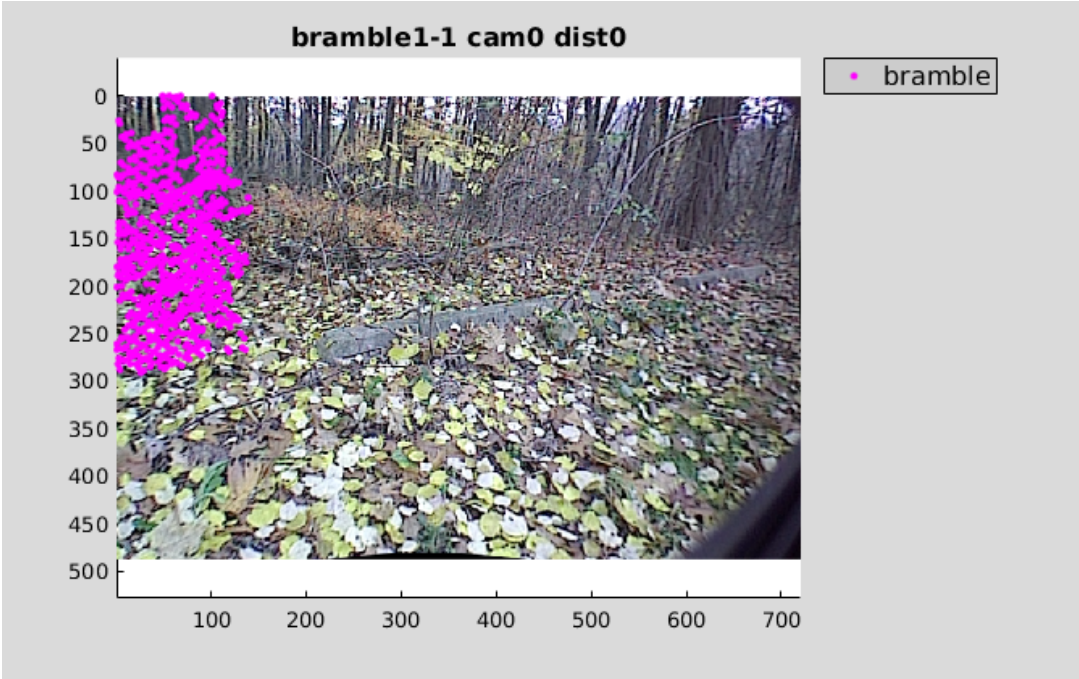


(a)

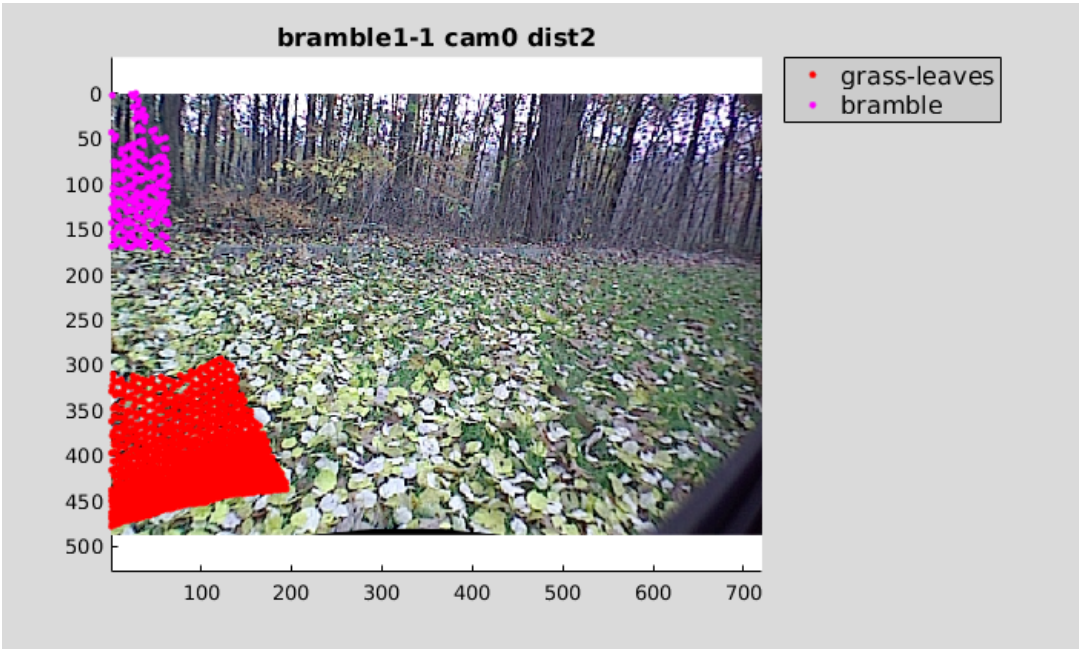
Figure 9.48: Self-Supervision Test Predictions (vt500-bumper, adxl-axle-up), Continued, Image 5



### 9.2.9 Vision Ceiling Benchmark Training Labels



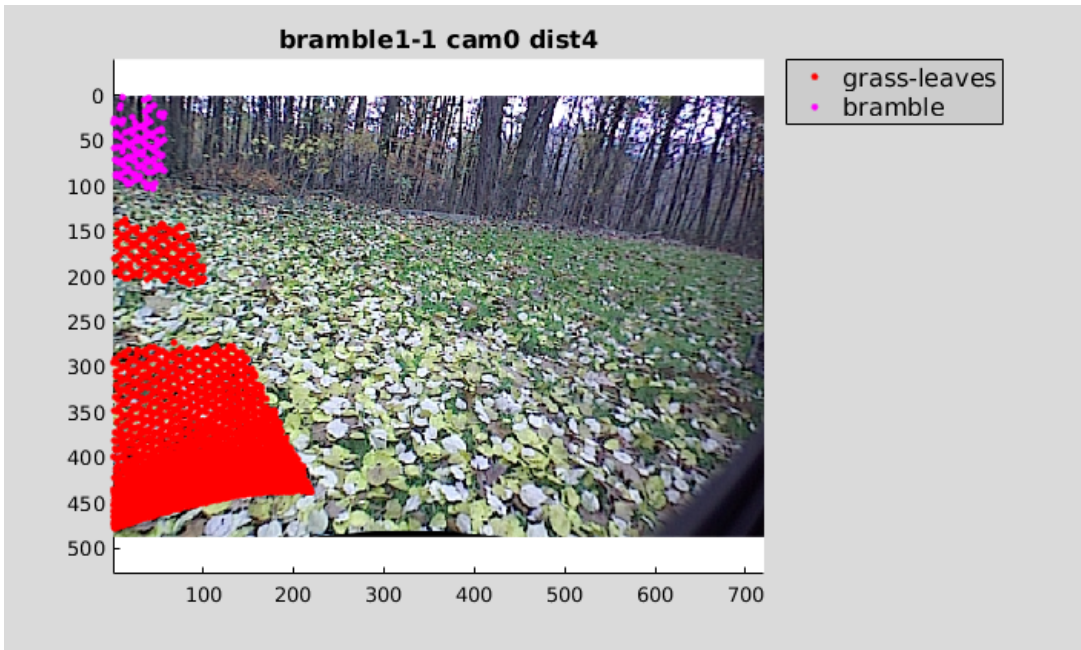
(a)



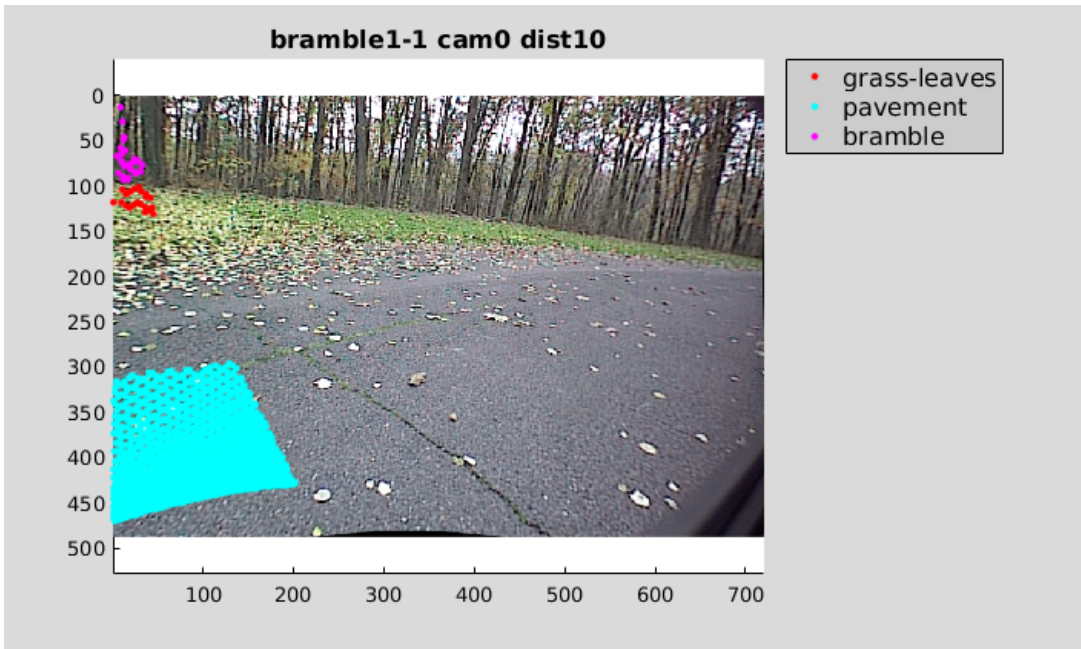
(b)

Figure 9.49: Vision Ceiling Benchmark Training Labels, Images 1 - 2. Colored points show human labels for supervised training data (locale bramble1-1, inside the robot's path). Refer to 9.2.3, 9.2.4, 9.2.5 for bare images. Refer to images 9.52, 9.53, 9.54 for subsequent predictions on test data.



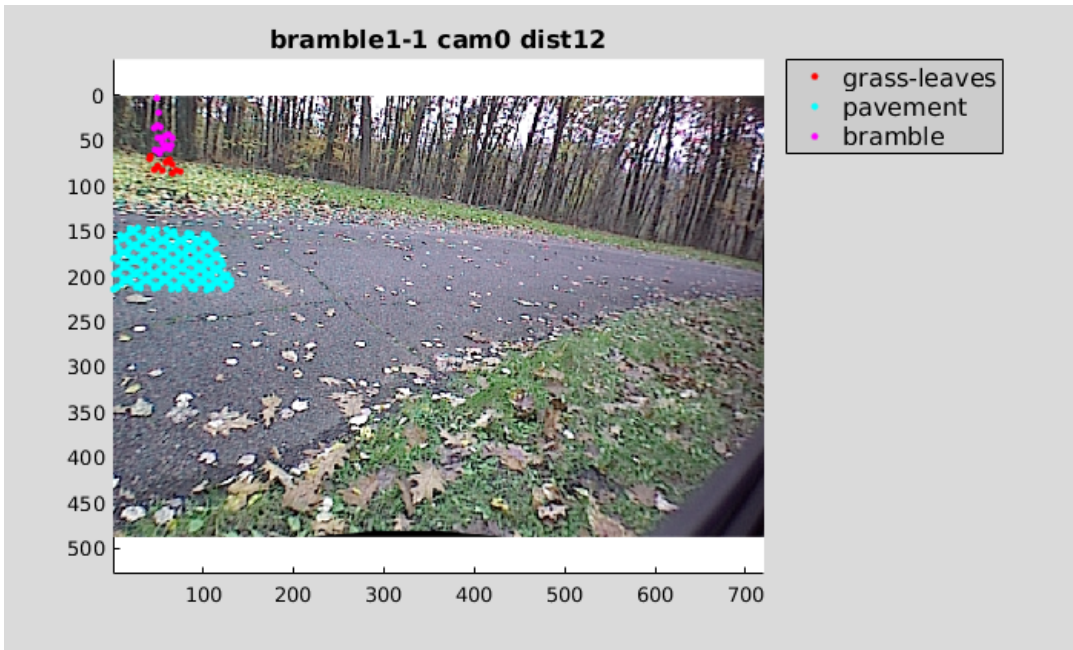


(a)



(b)

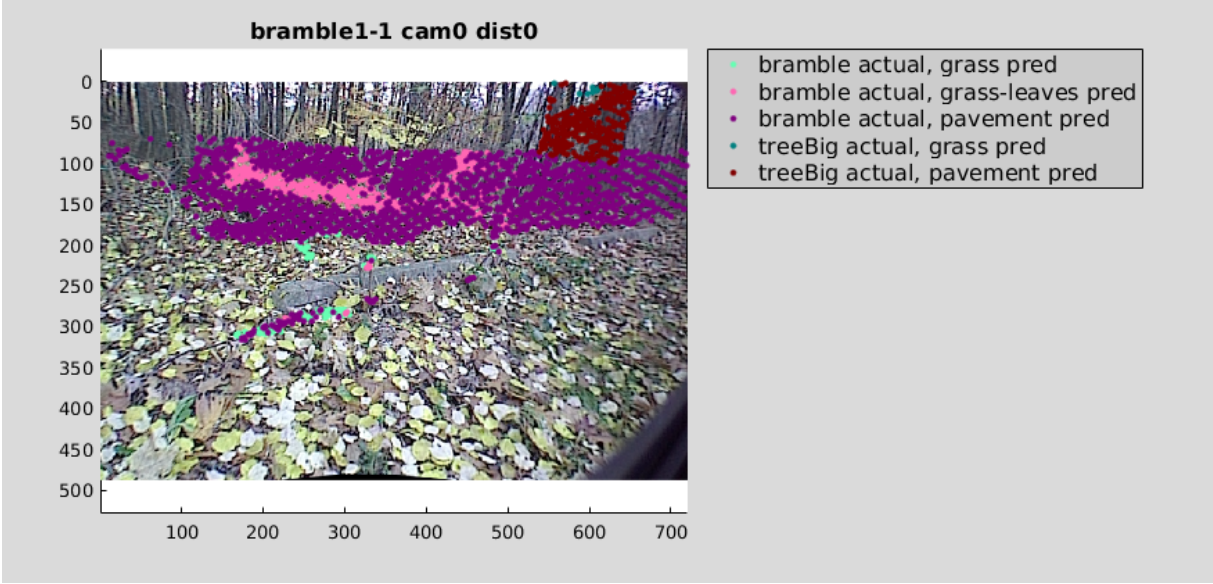
Figure 9.50: Vision Ceiling Benchmark Training Labels, Continued, Images 3 - 4



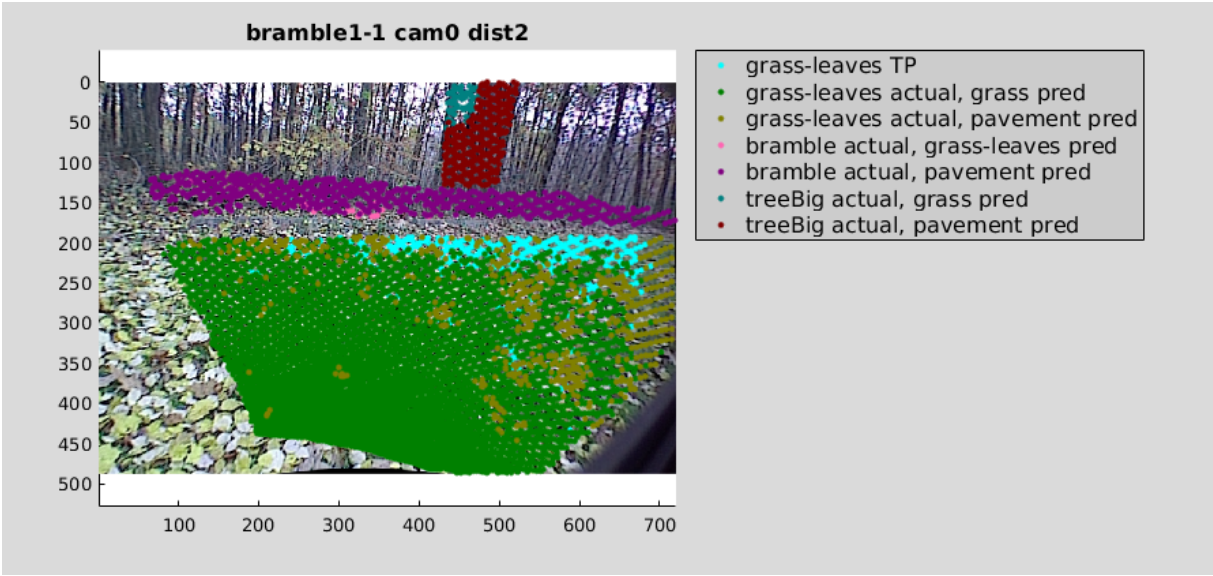
(a)

Figure 9.51: Vision Ceiling Benchmark Training Labels, Continued, Image 5

### 9.2.10 Vision Ceiling Benchmark Test Predictions

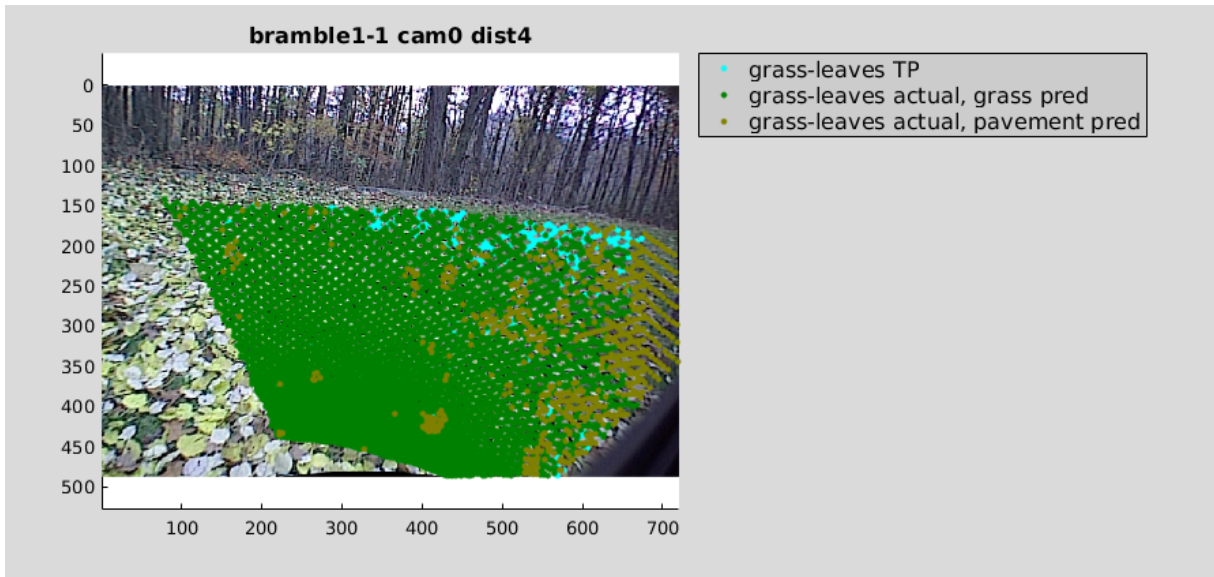


(a)

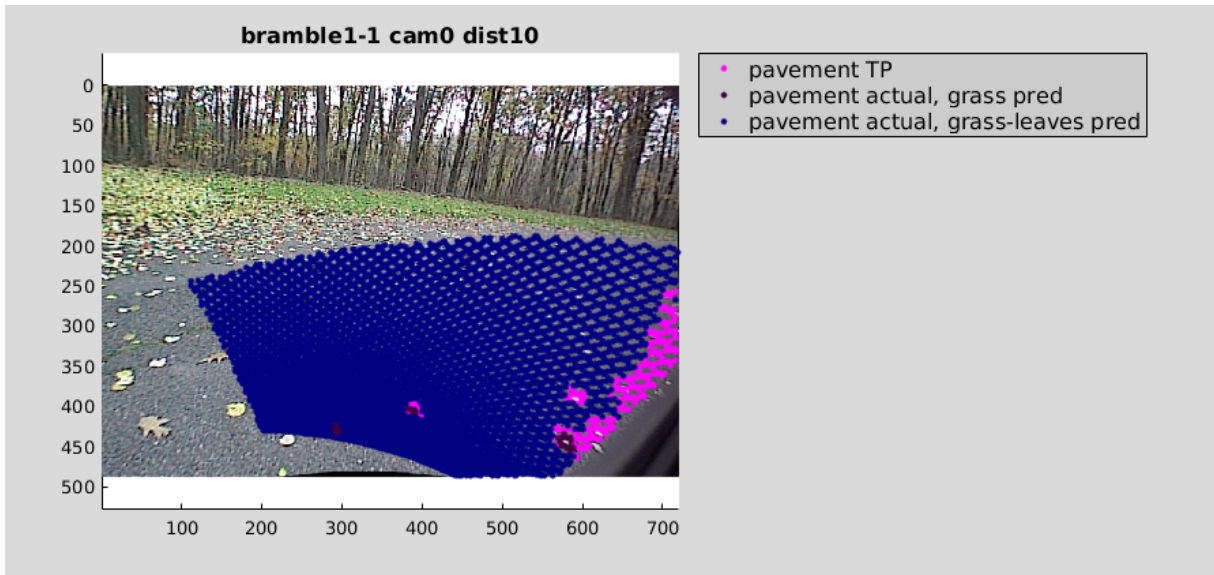


(b)

Figure 9.52: Vision Ceiling Benchmark Test Predictions, Images 1 - 2. Colored points show visual predictions from supervised classifier on test data (locale bramble1-1, outside the robot’s path). The model is trained on data from same locale, inside the robot’s path. Refer to 9.23, 9.24, 9.25 for bare images. Refer to images 9.49, 9.50, 9.51 for training labels.

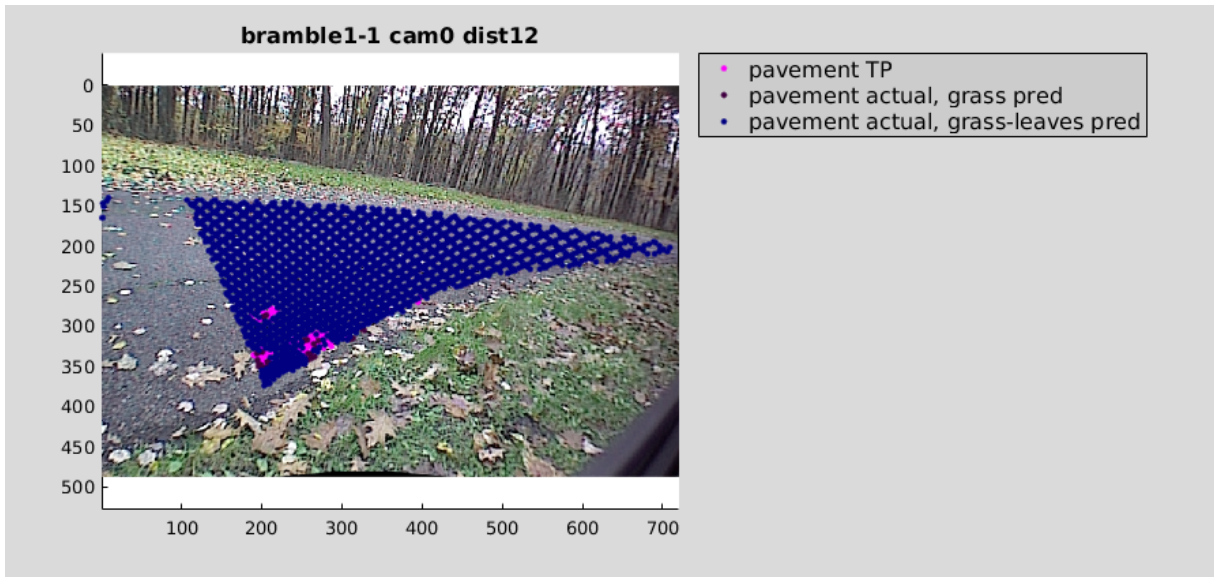


(a)



(b)

Figure 9.53: Vision Ceiling Benchmark Test Predictions, Continued, Images 3 - 4



(a)

Figure 9.54: Vision Ceiling Benchmark Test Predictions, Continued, Image 5



# Bibliography

- [1] Tensor Flow Libray, Convolutional Variational Auto Encoder. <https://www.tensorflow.org/tutorials/generative/cvae>. Accessed: 2019-12-02. 6.1
- [2] Aliza Amsellem and Octavian Soldea. Function-Based Classification from 3D Data and Audio. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2006. 5.1
- [3] Anelia Angelova, Larry Matthies, Daniel Helmick, and Pietro Perona. Learning slip behavior using automatic mechanical supervision. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2007. 2.2.2
- [4] Anelia Angelova, Larry Matthies, Daniel Helmick, and Pietro Perona. Learning and Prediction of Slip from Visual Information. *Journal of Field Robotics (JFR)*, 24(3): 205–231, 2007. 2.2.2, 2.3, 2.2, 2.3, 2.3, 3.2
- [5] Anelia Angelova, Larry Matthies, Daniel Helmick, and Pietro Perona. Dimensionality Reduction Using Automatic Supervision for Vision-Based Terrain Learning. In *Robotics: Science and Systems (RSS)*, 2007. 2.2.2
- [6] J. Andrew Bagnell, Felipe Cavalcanti, Lei Cui, Thomas Galluzzo, Martial Hebert, Moslem Kazemi, Matthew Klingensmith, Jacqueline Libby, Tian Yu Liu, Nancy Pollard, Mihail Pivtoraiko, Jean-Sebastien Valois, and Ranqi Zhu. An Integrated System for Autonomous Robotics Manipulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012. 5.5, 8.1
- [7] Max Bajracharya, Andrew Howard, Larry Matthies, Benyang Tang, and Michael Turmon. Autonomous Off-Road Navigation with End-to-End Learning for the LAGR Program. *Journal of Field Robotics (JFR)*, 26(1):3–25, 2009. 1.1, 2.2.1
- [8] Christopher A. Brooks and Karl Iagnemma. Self-Supervised Terrain Classification for Planetary Surface Exploration Rovers. *Journal of Field Robotics (JFR)*, 29(3): 445–468, 2012. 2.2.2, 2.2, 2.3, 2.3, 3.2, 3.3, 3.2, 3.2, 5.1
- [9] Christopher J. C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998. 5.4
- [10] Joshua Christie and Navinda Kottege. Acoustics based Terrain Classification for Legged Robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2016. 2.1, 2.1, 2.3, 8.1
- [11] Eric J. Coyle and Emmanuel G. Collins. A Comparison of Classifier Performance for

- Vibration-based Terrain Classification. In *26th Army Science Conference*, 2008. 1.1, 2.1, 2.1, 2.3, 3.2, 3.3
- [12] Robert S. Durst and Eric P. Krotkov. Object Classification from Analysis of Impact Acoustics. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1995. 5.1
- [13] I. Fogel and D. Sagi. Gabor Filters as Texture Discriminator. *Biological Cybernetics*, 61(2):103–113, 1989. 6.2, 6.3
- [14] Theodoros Giannakopoulos, Kosmopoulos Dimitrios, Aristidou Andreas, and Theodoridis Sergios. Violence Content Classification Using Audio Features. In *Hellenic Artificial Intelligence Conference*, 2006. 5.1, 5.3
- [15] Ashkan Hajjam. A near-to-far learning framework for terrain characterization using an aerial / ground-vehicle team. Master’s thesis, Mechatronics Systems Engineering, University of Denver, Denver, CO, January 2016. 2.2.2, 2.2, 2.3, 2.3
- [16] Derek Hoiem and Rahul Sukthankar. SOLAR: Sound Object Localization and Retrieval in Complex Audio Environments. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2005. 5.1, 5.3
- [17] Andrew Howard, Michael Turmon, Larry Matthies, Benyang Tang, and Eric Mjolsness. Towards Learned Traversability for Robot Navigation: From Underfoot to the Far Field. *Journal of Field Robotics (JFR)*, 23(11-12):1005–1017, 2006. 2.2.1, 2.2, 2.3, 3.2
- [18] Thorsten Joachims. Making Large-Scale SVM Learning Practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1999. 5.4, 6.2
- [19] Dongshin Kim, Jie Sun, Sang Min Oh, James M. Rehg, and Aaron F. Bobick. Traversability Classification using Unsupervised On-line Visual Learning for Outdoor Robot Navigation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2006. 2.2.1, 2.2, 2.3, 3.2
- [20] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *International Conference on Learning Representations (ICLR)*, 2014. 6.1
- [21] Eric Krotkov, Roberta Klatzky, and Nina Zumel. Robotic Perception of Material: Experiments with Shape-Invariant Acoustic Measures of Material Type. In *International Symposium on Experimental Robotics (ISER)*, 1996. 5.1
- [22] Jacqueline Libby and Anthony J. Stentz. Using Sound to Classify Vehicle-Terrain Interactions in Outdoor Environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2012. 2.1, 5.5, 8.1
- [23] Eric Martinson and Alan Schultz. Auditory Evidence Grids. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2006. 5.1
- [24] Ara V. Nefian, Luhong Liang, Xiaobo Pi, Liu Xiaoxiang, Crusoe Mao, and Kevin Murphy. A Coupled HMM for Audio-Visual Speech Recognition. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2002. 5.1
- [25] Lauro Ojeda, Johann Borenstein, Gary Witus, and Robert Karlsen. Terrain Charac-



- terization and Classification with a Mobile Robot. *Journal of Field Robotics (JFR)*, 23(2):103–122, 2006. 2.1, 2.1, 2.3, 3.2, 3.3, 8.1
- [26] Kyohei Otsu, Masahiro Ono, Thomas J. Fuchs, Ian Baldwin, and Takashi Kubota. Autonomous Terrain Classification With Co- and Self-Training Approach. *IEEE Robotics and Automation Letters, (RA-L)*, 1(2):814–819, 2016. 2.2.2, 2.2, 2.3, 2.3
- [27] Lionel Ott and Fabio Ramos. Unsupervised Incremental Learning for Long-Term Autonomy. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2012. 2.2.1, 2.2, 2.3, 8.2
- [28] Geoffroy Peeters. A Large Set of Audio Features for Sound Description (Similarity and Classification) in the CUIDADO Project, 2004. 5.3
- [29] John C Platt, Nello Cristianini, and John Shawe-taylor. Large Margin DAGs for Multiclass Classification. In *Advances in Neural Information Processing Systems (NIPS)*, 2000. 5.4, 6.2
- [30] Arturo L Rankin, Tonislav Ivanov, and Shane Brennan. Methods for Evaluating the Performance of Unmanned Ground Vehicle Water Detection. *International Journal of Intelligent Control and Systems (IJICS)*, 16(2):40–51, 2011. 1.3
- [31] Manuel Reyes-Gomez, Nebojsa Jojic, and Daniel P. W. Ellis. Deformable Spectrograms. In *AI and Statistics*, 2005. 5.1
- [32] Danilo J. Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. In *International Conference on Machine Learning (ICML)*, 2014. 6.1
- [33] David Stavens and Sebastian Thrun. A Self-Supervised Terrain Roughness Estimator for Off-Road Autonomous Driving. In *Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, 2006. 2.2.1, 2.2, 2.3, 3.2
- [34] David Stavens, Gabriel Hoffmann, and Sebastian Thrun. Online Speed Adaptation using Supervised Learning for High-Speed, Off-Road Autonomous Driving. In *International Joint Conference on Artificial intelligence (IJCAI)*, 2007. 2.1, 2.1, 2.3, 3.2
- [35] Abhinav Valada, Luciano Spinello, and Wolfram Burgard. *Deep Feature Learning for Acoustics-Based Terrain Classification*, volume 3. Springer International Publishing, 2018. 2.1, 2.1, 2.3, 8.1
- [36] Jacob Walker, Carl Doersch, Abhinav Gupta, and Martial Hebert. An Uncertain Future: Forecasting from Static Images Using Variational Autoencoders. In *European Conference on Computer Vision (ECCV)*, 2016. 6.1
- [37] Christian Weiss, Holger Frohlich, and Andreas Zell. Vibration-based Terrain Classification Using Support Vector Machines. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2006. 2.1, 2.1, 2.3, 3.2, 3.3
- [38] Carl Wellington and Anthony Stentz. Learning Predictions of the Load-Bearing Surface for Autonomous Rough-Terrain Navigation in Vegetation. In *International Conference on Field and Service Robotics (FSR)*, 2003. 2.2.1, 2.2, 2.3, 3.2
- [39] Mark C Wellman, Nassy Srour, and David B Hillis. Feature Extraction and Fusion of

- Acoustic and Seismic Sensors for Target Identification. In *SPIE Peace and Wartime Applications and Technical Issues for Unattended Ground Sensors*, 1997. 5.1, 5.3
- [40] Huadong Wu and Mel Siegel. Correlation of Accelerometer and Microphone Data in the Coin Tap Test. *IEEE Transactions on Instrumentation and Measurement*, 49(3): 493–497, 2000. 5.1
- [41] H. W. Yu and B. H. Lee. A Bayesian Approach to Terrain Map Inference based on Vibration Features. In *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, 2017. 2.1, 2.1, 2.3, 8.1
- [42] Kai Zhao, Mingming Dong, and Liang Gu. A New Terrain Classification Framework Using Proprioceptive Sensors for Mobile Robots. *Mathematical Problems in Engineering*, 2017(3938502):1–15, 2017. 2.1, 2.1, 2.3, 8.1