WatkeRR - Robotic Research Platform and Machine Vision System: Design, Development and Construction

by

Andrew McCormick

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Mechanical and Mechatronics Engineering

Waterloo, Ontario, Canada, 2018

Abstract

Rollators/wheeled walkers are a commonly used mobility aid that are utilized by people with a wide array of mobility challenges such as Alzheimer's disease, stroke, multiple sclerosis, and general loss of stability and control associated with aging or injury. The main benefit of walkers is that they are low cost, have a simple design, and allow users to maintain independence and continued physical activity. The largest concern with walker use is that accidents and improper use can cause falls. Falls can have serious repercussions such as broken bones, soft tissue damage, further loss of mobility, and increased fear. The focus of this research is a continuation of previous work focused on making walkers safer through understanding the use of assistive automation. Walkers with the addition of automation are termed Robotic Rollator (RR).

A significant amount of research has been already done on RR. Two main strategies for RR research is either modifying existing manual rollators or creating a custom RR from the ground up. Each option has its difficulties. The modified rollators have the problem that commercial walkers have such a wide variation in designs that sensor readings may not be transferable from one model to another model. From experience on an earlier iteration of the WatkeRR project, force and odometry measurements can be significantly affected by flexing and sliding of the lightweight frames. For these reasons it was decided that a custom RR would be used. This prompted the development of the Waterloo Robotic Rollator (WatkeRR). The WatkeRR was built to be a platform for measuring and testing low-cost controls and sensors. The base frame is designed to be able to match existing dimensions of most, if not all, commercial rollators by adjusting to match height, width, handle position, and wheelbase. It was important that the controls and data collection systems are untethered to allow for use outside a lab. To make the project more accessible, the controls are modular and built on an open source control structure.

The open source modular nature of the platform allows for parallel projects to be conducted at the same time. The following features are discussed: force sensing handles, modular caster wheels, wheel force loading, odometry, top-level human/WatkeRR control, Robot Operating System (ROS) integration/simulation, vision systems, and machine learning. Controlling the human/walker system is a challenging problem as the system can vary even during use. Change in the control of a walker is different for each user, including gripping, fatigue, injury, or walker type. The critical challenge of the control problem is the fact that the desired path of the user is not known. The data from a user can also vary depending on their physical and mental state. To be able to determine the desired path without direct input from the user three separate approaches can be utilized: 1) forces applied to the walker through the handles, 2) tracking position/pose of the user, and 3) detecting probable path to a known object. While the scope of determining user intent being larger than this thesis, a major focus will be on object detection. Determining what objects of interest are around a rollator allows for an insight into user intent.

The use of Convolutional Neural Networks (CNN), a deep learning approach that has shown promise for object detection is a field that has recently been advancing rapidly. This advancement has enabled safer and more effective autonomous vehicles. The second part of this thesis focuses on enabling object detection on the WatkeRR and integrating into a larger control strategy. The challenges of CNN application to the WatkeRR is that it needs to be achieved with fewer sensors, lower cost components, and less processing power. The navigation of WatkeRR can be performed indoors or outdoors which, due to highly variable lighting conditions, can cause problems with a vision system. As well, moving in crowds is not as much of a well defined problem as moving on roads with a car.

Only a few years ago, to train a CNN to detect even simple cases required days or weeks and access to a super computer. Now, top-level retraining on a commercial GPU can take a few minutes. This top- level training utilizes existing inception network structures. My work covers methods for obtaining training data, comparisons to a wide array of different input data as well as a hierarchical structure for improving precision. A significant amount of current work on CNN is focused on black box approaches that can determine a large number of classes. The disadvantage of this is that training takes significantly longer and detection accuracy for specific classes can be lower. With the improved training data and the application of the proposed hierarchical structure being used to train an established network. You Only Look Once (YOLO) used for detecting and localizing a wide range of objects.The thesis concludes with a set of recommendations on how to utilize this network in the ROS and how it will be integrated into path planning.

Acknowledgements

I would like to thank all the people that helped make this possible. Dr. James Tung, P. Eng and Dr. Jan Huissoon, P.Eng for constant guidance and support in supervising this project. All the fellow students, Mina, Alex, Jay, Seamus, David, Josh, Rick, Joey, Alex, Susan, and Wasim who have been a huge help in the WatkeRR project. Finally my family for providing encouragement and support during this time.

Dedication

This is dedicated to my grandparents Doris, Milton, and Maxine. I hope this work can help make assisted locomotion safer.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The current global human population is experiencing two major demographic shifts. The birth rate is declining and life expectancy is increasing. Increased life expectancy can be attributed to improved diet and health care and is seen as an indicator of improving quality of life. This will have the effect of increasing the global population. Declining birth rates will slow the exponential growth of the population and help counteract the effects of increased life expectancy. The peak growth rate was 2.1% per year in 1962. The current and expected changes can be seen in Table 1.1 [15]. Since 1962 the population growth rate has been decreasing but the people born during this peak continue to age. This will have a profound effect on society as these people leave the workforce.

Table 1.1: Population Past and Projected Demographics Changes [15]

	2000	2015	2030	2050
% Population Growth per Year	1.25	1.19	0.75*	0.47*
Number of Children Per Woman	2.66	2.5	2.25*	1.99*
Life Expectancy in Years	66.4	70.4	73.3*	83.3*
Millions of People Over 60 (%)	3.08(10.0)	4.37(12.2)	6.44(16.5)*	9.39(22.1)*
Millions of People Over 80 (%)	0.34(1.1)	0.61(1.7)	1.13(2.9)*	1.96(4.6)*

*Projected estimate

These effects are amplified in developed nations; in Canada the birth rate is currently 1.6 children per woman, 0.9 lower than global average. The average life expectancy is 82 years, 11.6 years longer than the global average. The current population over the retirement age of 65 is 5 million and will rise to more than 10 million in the next 20 years; approaching 1/4 of the estimated 41 million people expected to be living in Canada [4]. The average cost of healthcare per person under the age of 65 is $2 341 a year. By the age of 80 the healthcare cost increases to $20 387 per person per year [17]. Innovation in elder care will be needed to manage the increased demand and cost. The effect of losing 5 million people from the work force to aging will strain an already strained system.

This is a large problem and a lot of research is being done in this broad field but more will need to be done to address this challenge. Our Bio-Mechanical research group is focused on finding solutions to mobility challenges through the use of low cost sensors in concert with advanced automation. Of the 3.8 million people classed as disabled in Canada 7.2% have mobility related disabilities[4]. For people over the age of 65, mobility impairments increase to 20.6%[2]. To be classified as having a mobility disability, the respondents have persistent difficulty walking for 15 minutes on a flat surface or up and down stairs without an aid.

This can be caused by pain, weakness, injury or ailment which can result in reduced locomotion, stability, balance, and/or endurance. Two of the most common assistive devices used by the elderly for mobility are wheeled walkers/rollators, and wheelchairs.

Wheelchairs can be used for almost any level of impaired mobility. This advantage comes at the cost of lost exercise by the user, risk of bed sores and decreased independence. Loss of independence can lead to further physical and mental problems.[4] Motorized wheelchairs can give this independence back but are typically expensive, have a limited range due to the electric battery and further reduce physical activity in use. Increasing the amount of time a walker can be used before switching to a wheelchair can overcome these problems. The user of a rollator has been shown to increase movement speed by over 200% and reduce side to side sway by 40% - 60 % [19]. Walkers can provide stability support with the advantage of physical activity as users go about their daily activities. Even though rollators are shown to increase mobility in users, approximately 50% of users quit using them shortly after initial use [5]. The primary reason for ceasing use is that users find them difficult to operate. Furthermore, rollators have a large footprint that can make maneuvering though doors, around obstacles, and in tight spaces very challenging.

Trying to push a rollator through a door that is being held open by the user is no simple task, especially for someone with reduced mobility. An even bigger concern is that the use of rollators has been associated with an increased incidence of emergency room visits due to falls [3]. Users of walkers identified that they had experienced falls 83% walking down hill ,77% uphill, and 73 % maneuvering through doors [9]. The cause of falls have four major sources: collision with a solid object like a door frame or curb; the walker rolls out of the users reach; the transition from standing with the walker and sitting on a chair, and the user forgets to set the brake when using the walker as a chair. Falls in the elderly can be life-threatening and can cause large changes in quality of life. As a result, the persons mobility and social interaction can become limited by the fear of falling.

Studies have shown that a fear of falling has been found to increase as a person ages. This fear can significantly reduce a person's quality of life when they can no longer complete everyday tasks. Not being able to do simple daily tasks can also cause further health problems, in the form of muscle loss, malnutrition, and depression [4]. To be able to change this and keep the elderly moving safely, two things need to happen with mobility aids. They need to be simple to use and the user needs to have complete confidence that the device will keep them safe. The research discussed in this thesis focuses on addressing safety by reducing falls and increasing user confidence in the use of RR

Work on improving RR has been broken into support/safety improvement, user assistant, and sensor based assessment. The user assistance is broken into either passive or active. For active assistance most current inputs are broken into direct directional control (e.g. a joystick, directional buttons) or navigation with a map based control with users preferring the map based navigation with 3 out of 4 users [20].

The primary objective of the thesis is to build a platform to advance research and testing low cost RR. Chapter 2 features the construction and design motivation of the WatkeRR. The WatkeRR is a RR designed to be used to develop and test low-cost robotics on an open source platform. The intent is to be able to use this platform to bridge the gap between research and production devices, allowing safety features to real users. The driving impetus for the construction of the WatkeRR was testing done on the previous rollators modified for data collection. This provided inconsistent results with significant amounts of noise. The idea that a sturdy frame built with data collection in mind would provide more useful output to be used for analysis and control. The construction of WatkeRR allows for a wide array of projects to be done on the RR, such as studying the user interface, user intent detection, control, path planning and many more. In the scope of the second part the objective was to look at challenges of user intent. The biggest challenge with user intent seemed to be that the path planning algorithms could avoid obstacles but not determine what the user wanted to do. The remainder of the thesis is focused on machine learning to find objects people interact with and methods to make this more efficient and accurate.

The classification of images and preparation are covered in Chapter 3. Chapter 4 covers assessment of the data classified in Chapter 3, as well discussing the validity of the data. Better performing data sets are used in a Darknet detecion network to identify and locate objects in real time that a walker user would interact with in their day to day lives discussed in Chapter 5. Finally Chapter 6 covers conclusions that can be drawn from this work and discuss the array of future projects that can build on this work.

Chapter 2

WatkeRR

2.1 Introduction

Rollators have been used as a mobility assitive device for the past sixty five years. The rollator assists mobility by allowing users to transfer weight to the ground through the rollator which provides a wider base of support, resulting in improved balance. The initial addition of wheels and brakes improved the usability over the initial version that needed to be lifted by the user each step. Further design improvements and features have focused on making them lighter and collapsible for transport [12]. Improving walkers by automating them has been researched for more than thirty years. In that time, there has been significant progress in areas of user monitoring, force sensing, mapping, locomotion and path planning [11]. Despite this research robotic systems have not been implemented on rollators for use by the general public.

2.1.1 Robotic Rollators

Some common features that make a rollator a RR are controlled steering, sensors for detecting obstacles, and sensors for detecting the current user state. [6] These systems have been shown to be able to reduce potential falls by reducing collisions [18]. The focus of determining what the current state of the RR user (e.g. stationary, right foot stance, left foot stance, push) have been extensively studied [7]. Methods for tracking what the user is doing are broken into several main approaches:

- Walking event detection

-Image processing from camera data.

-Lab based ground pressure plates.

- Force applied to rollator by users

  -Sensor with 6-degrees of freedom connected to handle.

  -Stain sensors on handles

  -Load cells mounted in the handles.

  -Load cells mounted in the wheels.

- Ultrasonic range sensors measuring distance of users.

These methods can be combined to give a more complete picture of what is happening in the system [20]. Combining stationary motion capture with on-board distance sensors, Red Green Blue (RGB) cameras, 3D cameras and force sensors. These methods are useful for determining what the user is currently doing, but fall short when trying to predict what the user will do in the future.

A wide range of sensors are used for mapping however, an automated steering system requires a means to use that information in a way that helps the user and ensures safety. The difficulty is that the steering system has typically made the Rollator more complicated to use. Current RR have shown good results in using collision avoidance algorithms but have very little improvements in helping a person negotiate difficult obstacles like manually operated doors. The purpose of this work is to create an automated method to reliably identify objects that a RR user could need help interacting with and be able to use that information in a way that helps the user. This will improve safety and confidence of the RR user. To keep the operation of the RR simple, the input will just be the user pushing as they would a non-robotic rollator, while the RR still provides control.

2.1.2 Research at University of Waterloo

Our research group at the University of Waterloo has produced several modified walkers with various features. The rollator shown in Figure 2.1.2 in the left panel is a modified standard rollator. It has been modified to allow for automatic activation of the brakes when the user stops walking and deactivates them when they go to stand up. This was done with pressure sensing insoles for shoes with a wireless connection to a custom braking system. The purpose of this design was to limit falls from users failing to set the brakes when they switch from walking to sitting on the walker.

The walker shown in the right panel of Figure 2.1.2 has strain gauges mounted on the frame and load cells mounted in the handles. The group modified the walker to detect if the user was holding on to the handles. My initial research was to see if this sensors configuration could be used to determine user intent. In initial testing of strain gauge sensors mounted on the frame, readings were found to give binary information that the walker was being interacted with but was not very detailed. This could be used for braking or to determine a change in direction of the walker. The load cells on the handle were found only to provide readings when gripped directly over the sensors. This was found to be too inconsistent to provide data for determining user intent.

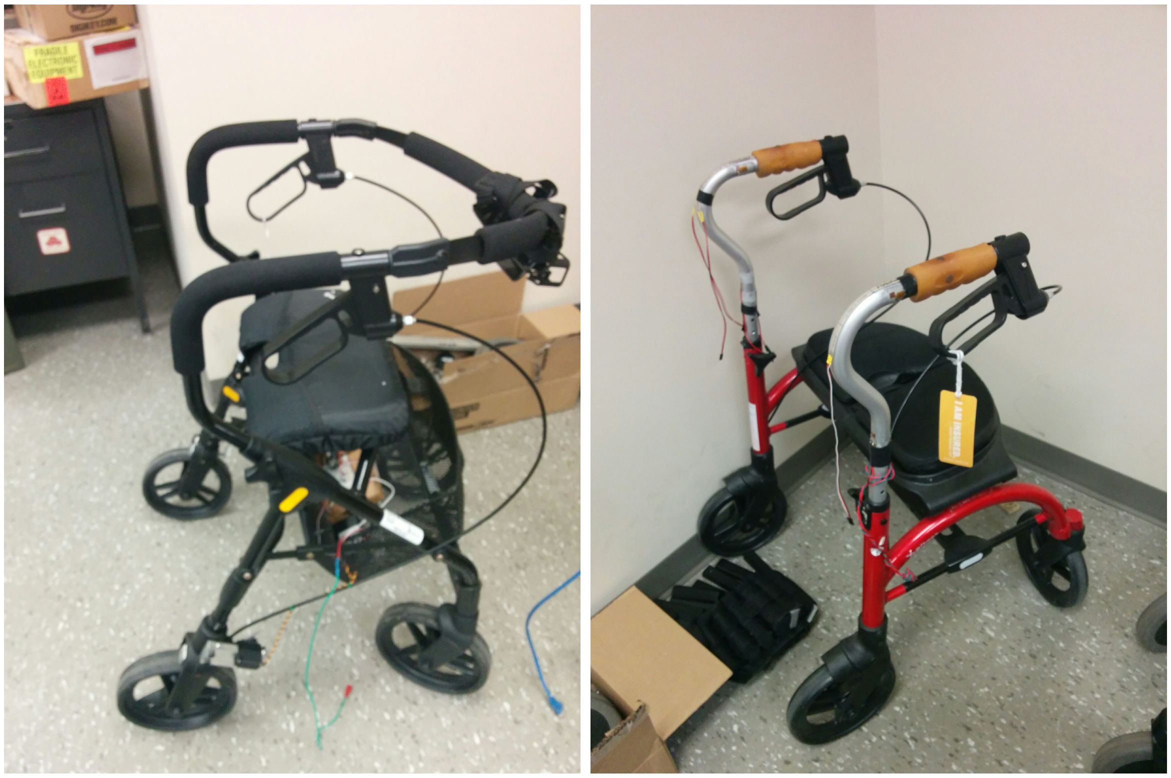


Figure 2.1: Walkers modified with automatic braking(left) and force sensors(Right)

The walker shown in the left panel of Figure 2.1.2 is a manual rollator converted to a RR by adding a Kinect $^{\text{TM}}$ camera, braking to the back wheels and using a computer for control. The Kinect $^{\text{TM}}$ is an off the shelf product made for the Microsoft Xbox 360 gaming console. It is a powerful, low cost sensor that has a 3D colour camera. The camera was used to build a local 2D map and apply braking to assist users in avoiding objects. In early trials this system was able to reduce collisions over unaided motion. [18] This would then reduce falls caused by the sudden stop of the walker when it hits a solid object. The data processing and collision avoidance control was all programmed in Matlab for this specific task.

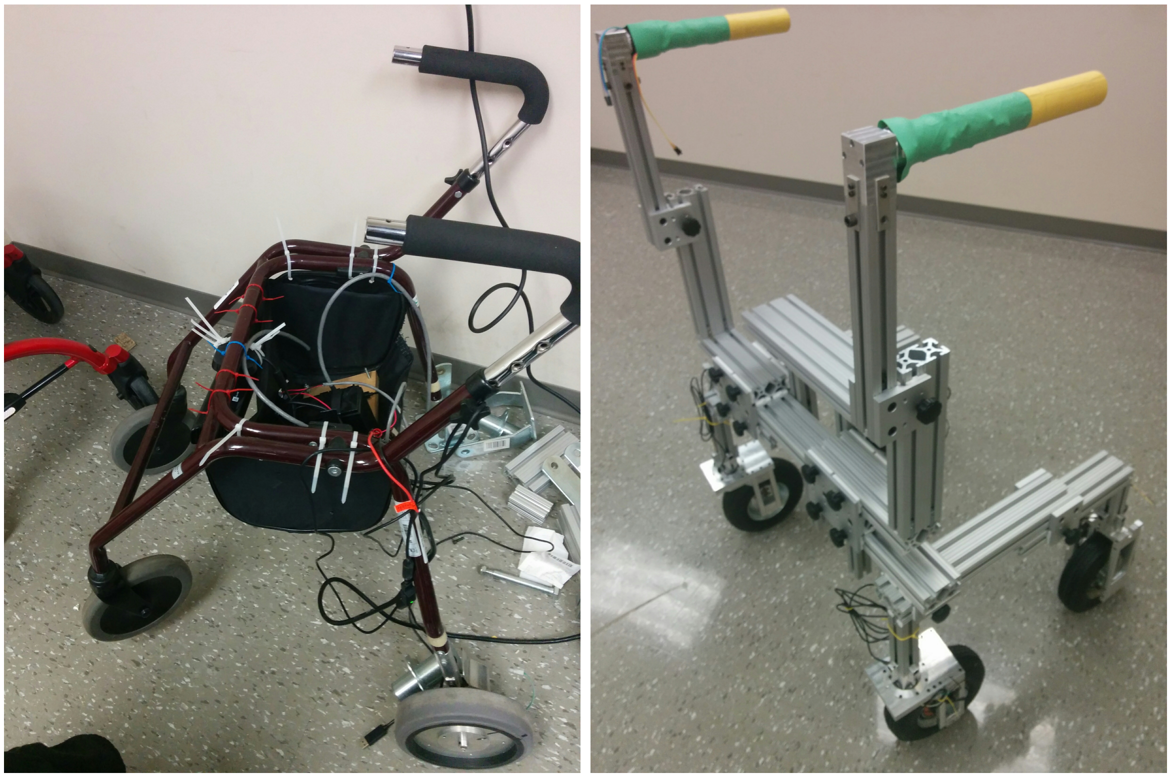


Figure 2.2: Robotic Rollators the Walker with 2-D mapping and automatic braking(left) and WatkeRR(right)

2.2 Design Features

Initially working with modified rollators provided significant challenges in data collection and readability. This was cause by variations in sizes and lack of consistency of shape and structure as well as flexing in the frame during load. This was the driving factor in building a new research platform. The intent with the platform is to get highly repeatable and representative data. Being configurable and modular are important for quick and easy development and quick design changes. The frame than can be used to test advanced controls, accurately measure user intent, and develop new vision systems. The system also needs to be able to operate wirelessly and independently outside of the lab. The design features that the WatkeRR were build on are:

- Standardized control and sensor integration with an open source platform. The system we used is called ROS.
  - Simulation software called Gazebo that is integrated with ROS for model based testing of complete system.
  - SLAM using a Kinect V2.0 for 3D mapping and path planning.
- Powered locomotion, steering and regenerative braking.
- Size and position of the frame that is fully adjustable to allow for quick and immediate matching of users existing rollators.
- Force sensing handles and wheels.
- Low cost smart vision system that has object detection and integrates with the mapping and path planning.

2.3 Hardware

2.3.1 Frame

The frame is built using off the shelf parts from 80/20 15 series and extruded aluminum. The frame construction allows for adjustment of the size of the key parameters of the walker: the wheel base, length and height/width/position of handles. This is done to allow the WatkeRR to be matched to ideal or the current configuration of the walker that the patient is using. In a review of existing walker sizes, the width ranges from 0.5588 m to 0.7112 m and the handle height range is 0.7112 m to 1.016 m. The WatkeRR has the width range 0.57 m to 0.77 m and handle height range of 0.86 m to 1.0 m. The frame with the wheels and handles mounted can be seen in Figure 2.3.

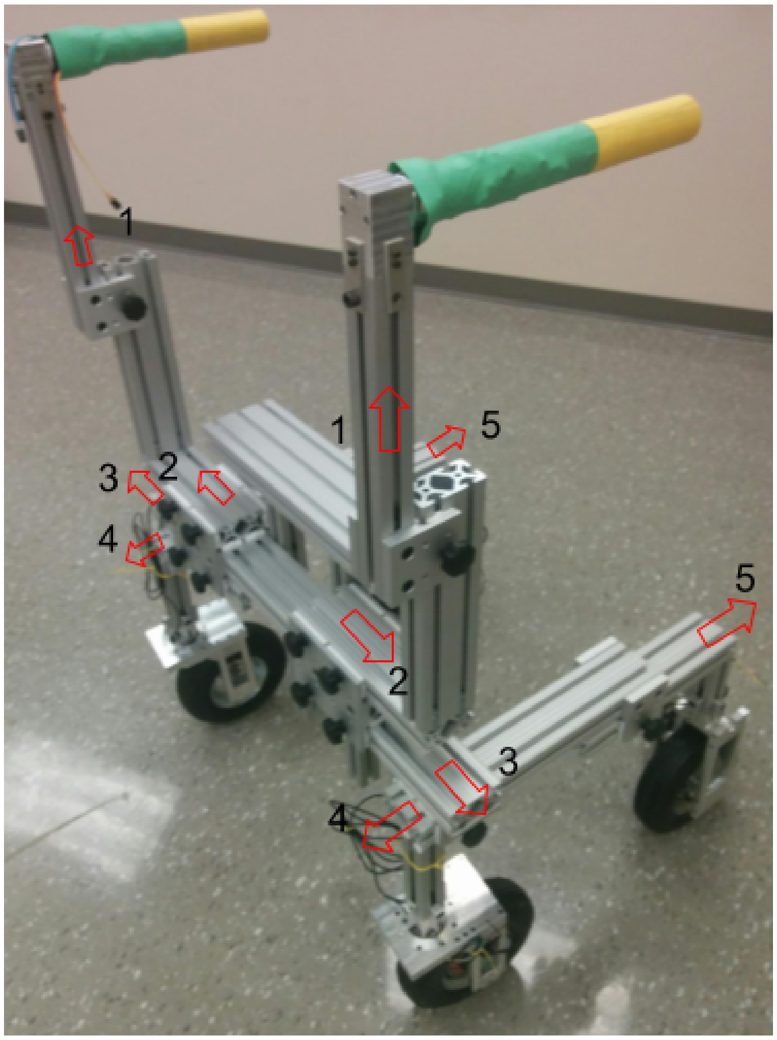


Figure 2.3: Isometric View of WatkeRR Showing Extensions

2.3.2 Handles

Two custom handles shown in 2.4 were built to measure applied forces. The forces measured are down force, side to side, and grip. The mounting location for the strain gauges is shown in 2.4. The handle being a straight cylinder that is cantilevered, any forces being applied cause a bending moment that is read by top and bottom strain gauges for downward force. Two side gauges measure lateral forces. The two slots cuts create cantilevers that create a differential on the strain gauges when grip force is applied by the user.

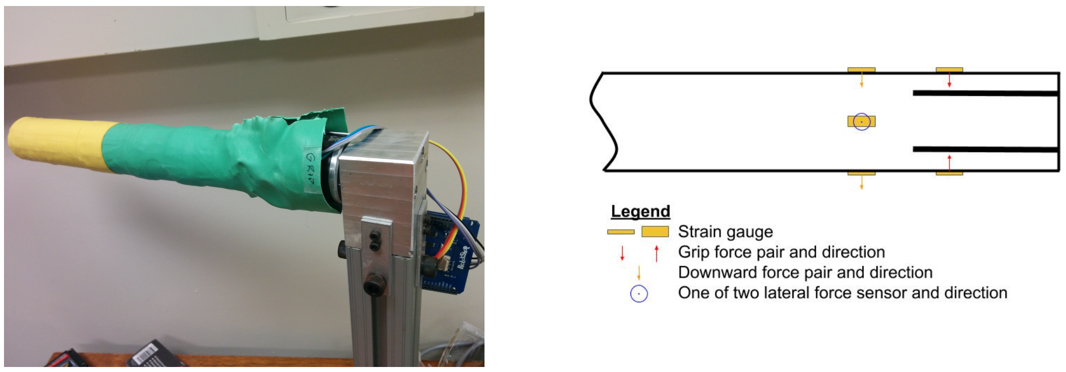

Legend
Strain gauge
Grip force pair and direction
Downward force pair and direction
One of two lateral force sensor and direction

Figure 2.4: Cantilever force sensing walker handle (Top), Strain Gauge locations(Bottom)

2.3.3 Wheels

The wheel design of the WatkeRR was built based around the basic idea of existing rollator wheels with two casters on the front and two fixed rotation wheels in the back. For simplicity, the existing diameter of eight inches was used but instead of a solid plastic/rubber wheel a wider inflatable rubber tire was used. This was done to increase traction and to reduce slip during driving, braking and encoder readings.

The design intent was to allow the user to still steer by pushing the walker in the desired direction. To enable control both powered steering and powered drive/braking were considered. Stepper motors will be used for powered steering. The output of the stepper will be geared up to increase torque. To be able to push the walker and autonomously steer, a clutch will be used to decouple the stepper from wheel.

The options for the powered wheel was either a geared motor or a motor set in the hub of the wheel. The geared motor has the advantage of increased torque but would require a clutch to allow free pushing. An in-hub motor needs to be a higher torque motor but has the advantages that it allows unrestricted rotation when not powered which means it will fail as a manual rollator. In-hub also has the potential for regenerative breaking. The application of regenerative braking could increase the life of the WatkeRR and potentially be used to provide resistive training to users.

The first iteration of the wheels shown in 2.5 was to have the front caster wheels be powered driving and steering. The back wheel would then just have the encoders for optometry. The key to this design was using a rotating connector to allow for the drive wheels to be powered as they rotated. Difficulties in implementation of the rotating connector and the drive gear with encoder lead to the design being replaced with a newer iteration of the design shown in 2.3.3.

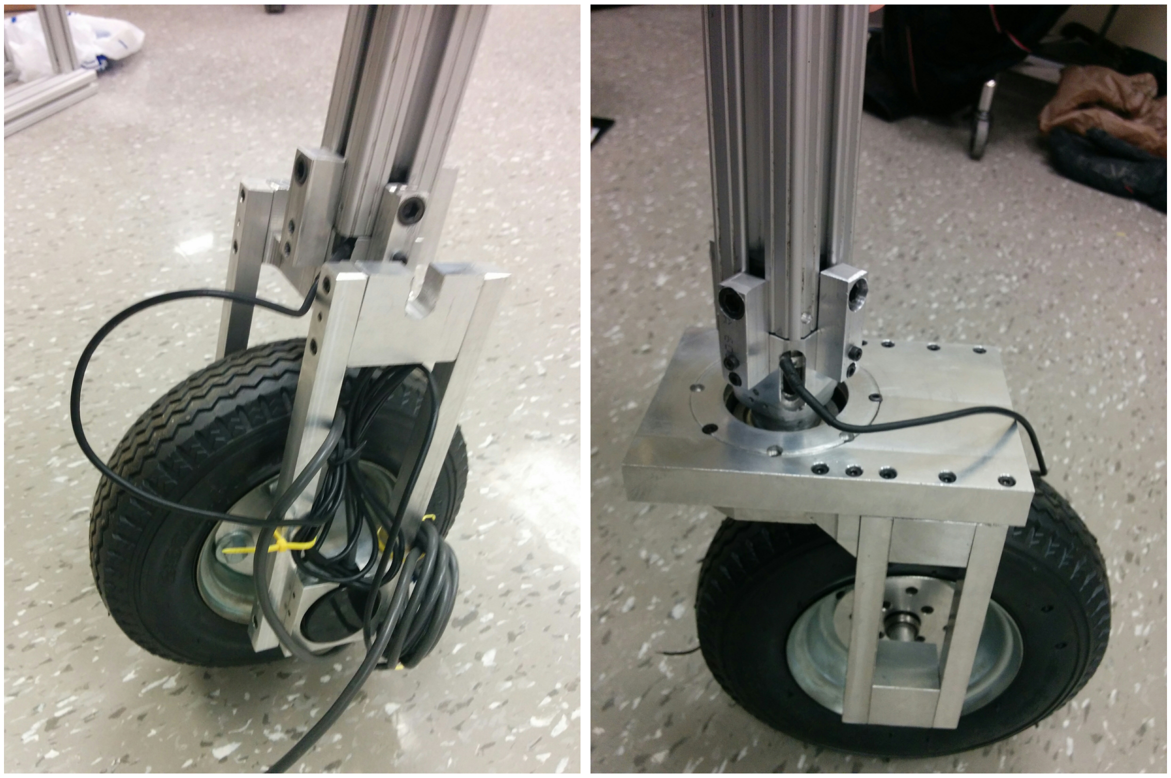


Figure 2.5: Initial design of WatkeRR modular wheels.

An intermediate step was used before the switch to the second iteration for data collection to be used in building the top level control models. The intermediate step moved the motors to the back wheels and the encoder to the caster wheel. This had the challenge that the direction the wheels were facing was not known. A permanent solution would be done with the second iteration, but due to time constraints a temporary solution was developed. This solution was to mount a mouse optical sensor to the back of the caster to track its movement. This was found to be reasonably accurate with less than a degree but can be susceptible to bias. The latest iteration of the modular wheels was built on

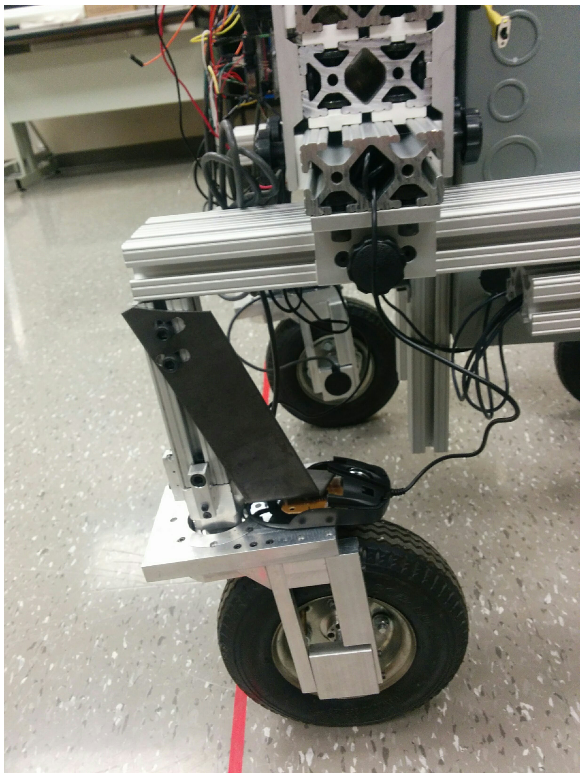


Figure 2.6: Temporary wheel tracking using an optical mouse for the WatkeRR modular wheels.

the frame work of the first wheels. The motor was moved to the rear wheels as part of the intermediate work. This had the benefit of simplifying the control and removing the need for the rotating power coupler. The improved design of the caster wheels is lighter and cleaner but also incorporates two 4096 tick/rev encoders for the angle and rotation of each wheel. The encoder as well as a stepper motor with clutch is connected to the wheels via timing belts.

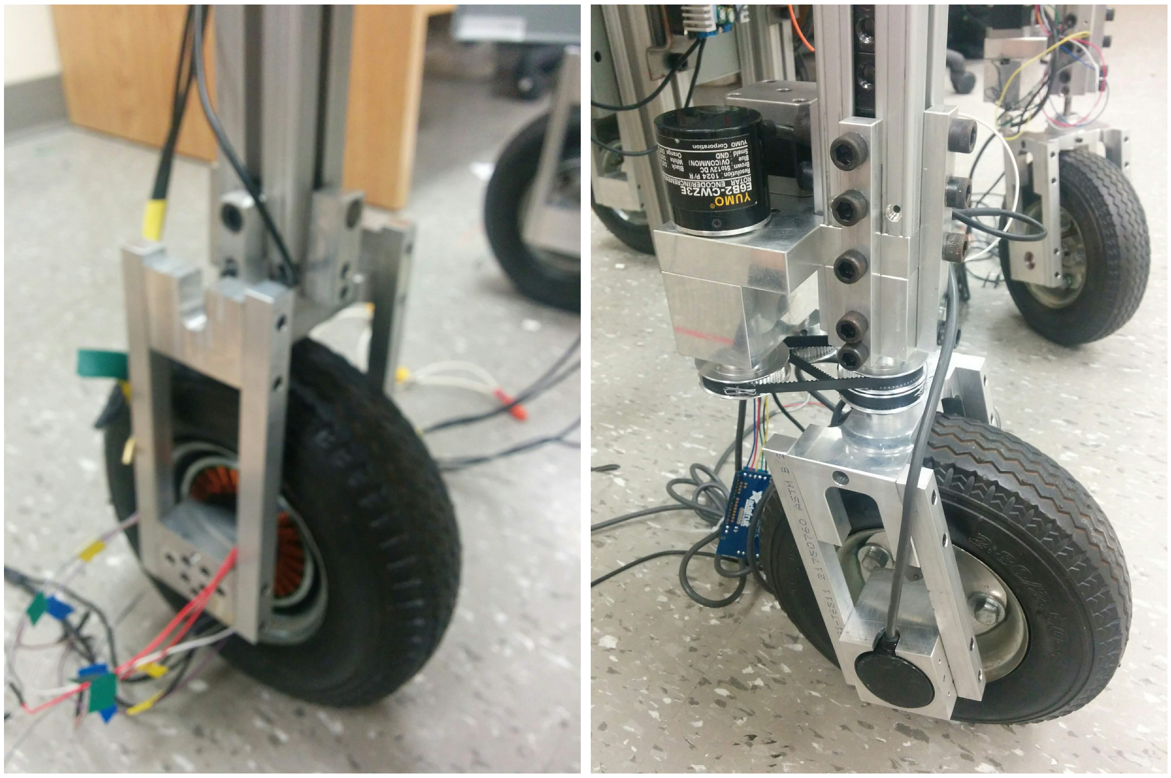


Figure 2.7: Current design of WatkeRR modular wheels.

2.3.4 Cameras

An important aspect of the WatkeRRsystem is the machine vision. The quality and ability of machine vision data is heavily determined by the quality of the data that comes into the system. Recent innovations in camera technology have significantly reduced the cost at the same time improving the quality of output images. The types of cameras that were looked at for the WatkeRR where infrared depth cameras, 360° field of view, and development board mounted csi-2 cameras. Beyond quality of image, the other important factor is the latency of data. For the images to be actionable by a control system the images need to go from camera sensor to being processed as fast as possible. For this application a latency of 200ms or less should be required . For the infrared depth camera a Kinect v2 was selected to be integrated into SLAM from the ROS) package. Since the initial selection Microsoft has discontinued support for the Kinect as a research tool and it may be necessary in the future to switch to another sensor such as a Real Sense Camera from Intel. With the current version of ROS being used the Kinect has been successfully integrated into the ROS environment using Real-Time Appearance-Based Mapping (RTAB). The kinect v2 provides a depth image in a range of 0.5 m to 4.5m. This can be recorded at 30 frames per second in a field of view of 70 by 60. This is good but for our application being able to find objects of interest outside of the window is important, so a camera with a wider field of view was selected for the machine learning part of the project.

Table 2.1: List of cameras and specifications tested for use in the WatkeRR Real-Time Appearance-Based Mapping project

Camera Name	**Resolution**	**Field of View**	**Frame Rate**	**Latency**
Kinect V2	1920 x 1080	70° x 60°	30 fps	72.98ms - 150ms
Giroptic	2048 x 1080	360° x 300°	30 fps	5000ms+
Go Pro	3840 x 2160	94.4° x 122.6°	30 fps	n/a
R-Pi Camera	2048 x 1080	180° x 180°	30 fps	100ms
Jetson TX2 camera	3840 x 2160	45° x 45°	30 fps	70ms - 100ms

To establish a large area of awareness the viability of using a 360° field of view Giroptic 360° camera was assessed. The high resolution video was adequate for training but the latency was over 3-5 seconds which was far too long for this application. The necessity to wait for custom software to stitch images was likely to be a latency bottle neck across all 360° cameras. To overcome the latency issues and to still allow for a full field of view pi-camera integrated into a RPi were assessed. The latency was found to be close to 100 ms from the multiple seconds of the 360° camera but to be able to get the same field of

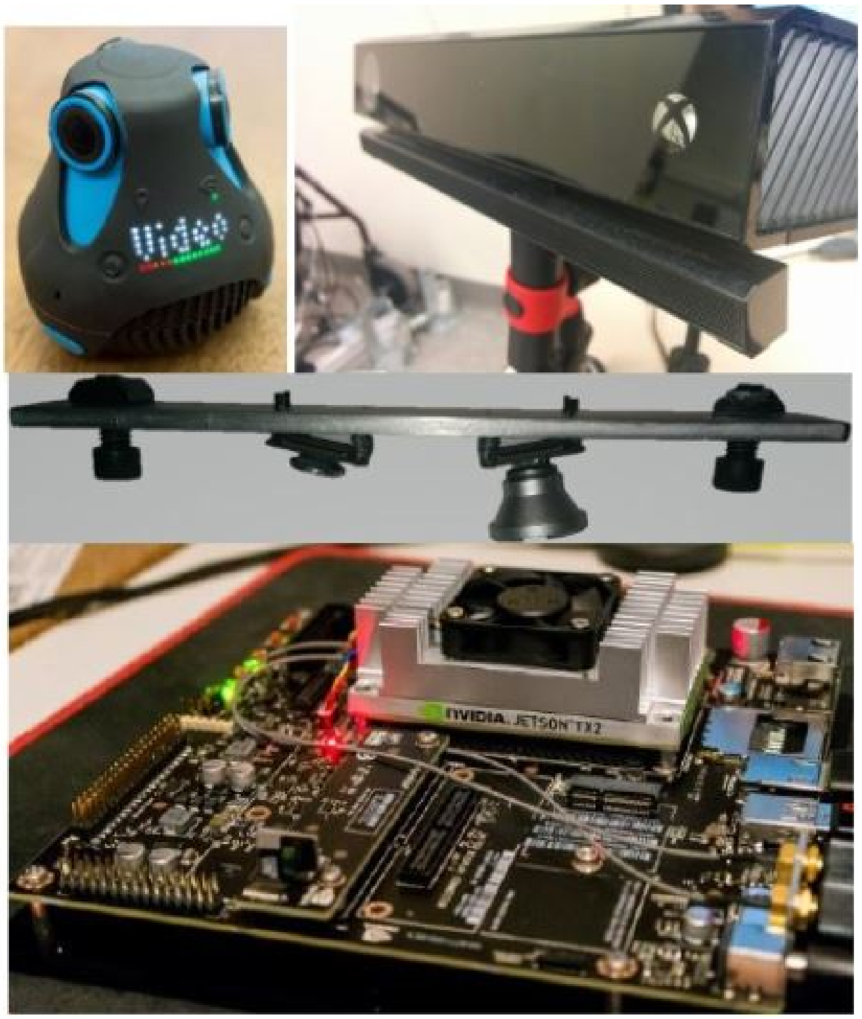


Figure 2.8: Giroptic 360° camera(top left), Kinect (top right),RPi cameras mounted with and without lens(middle), jetson tx2 with camera.(bottom)

view multiple cameras are needed with a high field of view. A 180° field of view lens was selected to be used as part of a two camera system. To be able to use the two cameras a multiplexing shield was used to be able to bring the data into the Pi. The shield is able to read up to four cameras at the same time so that a third camera could be added in the future to make a full 360° view.

The current configuration allows for approximately 270° field of view with overlap in the middle. The Pi cameras were used to record data for training the CNN. The training will not be done in real time but the processing will be. Recently, NVIDIA has made a development computer called a jetson. The second iteration the TX2 is a Linux based embedded computer that has capability to use the same cameras as the Pi but it can also do the training since it has 192 parallel processing cores (Compute Unified Device Architecture (CUDA)). The on-board camera was used to track the user of the WatkeRR.

2.3.5 Arduino

With the WatkeRR being setup as a test platform, it was built such that each system was separate to allow multiple people to work on the project at the same time. The basis of this is that each project is built on the low cost open source Arduino controllers. The Arduino used for this project are Bluno from DFRobotics. The advantage of the Bluno is that they have built-in Bluetooth and as a result are able to be programmed and can stream data wirelessly. We use the Mega and Uno versions of the Bluno; the Mega is shown in 2.3.5 as well as the WatkeRR with the Arduinos mounted.

2.3.6 On-board Computer

The system was designed then to be able to integrate all of the sensors with an on-board computer as shown 2.3.6. This computer was also used for training of the neural networks when integrated with a NVIDEA Titan Xp (TitanXp). The On-board computer has Matlab with Simulink and ROS Indigo installed with Ubuntu 14.04 as the operating system. The ROS workspace is built with modified existing packages. This computer adds processing power for real time data collection that would not be needed in a commercial version but is necessary for studying user interaction, developing control algorithms, and training neural networks.

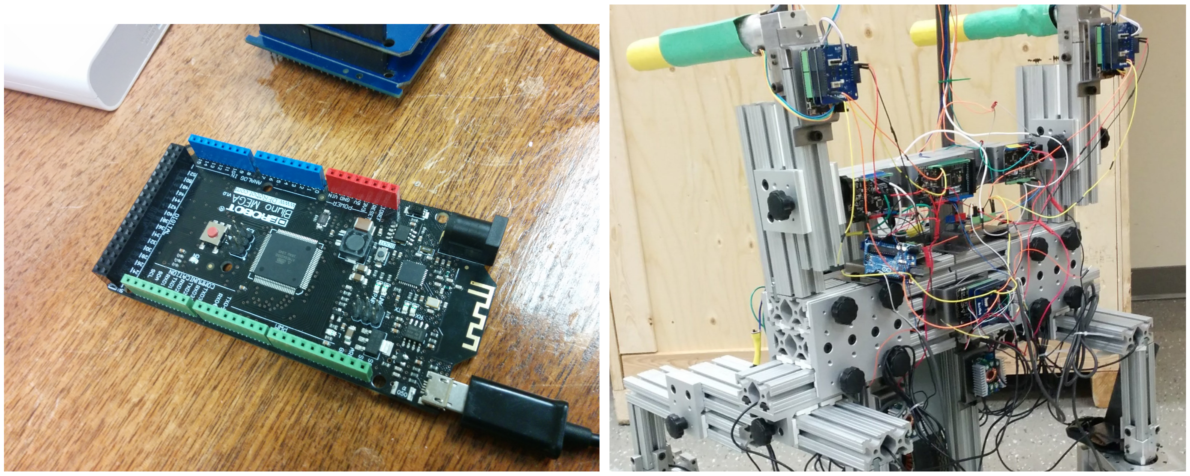


Figure 2.9: Arduino Mega Bluno Version(left), WatkeRR with Arduinos mounted(right)

Table 2.2: List of Arduinos currently used in project.

Board	**Function**	**Sensors**
Mega	Data combination and streaming	
Encoder Left	Wheel rotation/angle of left wheel	2x encoder
Encoder Right	Wheel rotation/angle of right wheel	2x encoder
Motor Control	Wheel speed and angle control	2x ESC 1x stepper controller
Wheel load	Force applied to each wheel	2x, load shield 4x button load cell
Handle left	Forces applied to left handle	2 x load shield 6x strain gauge
Handle right	Forces applied to right handle	2 x load shield 6x strain gauge

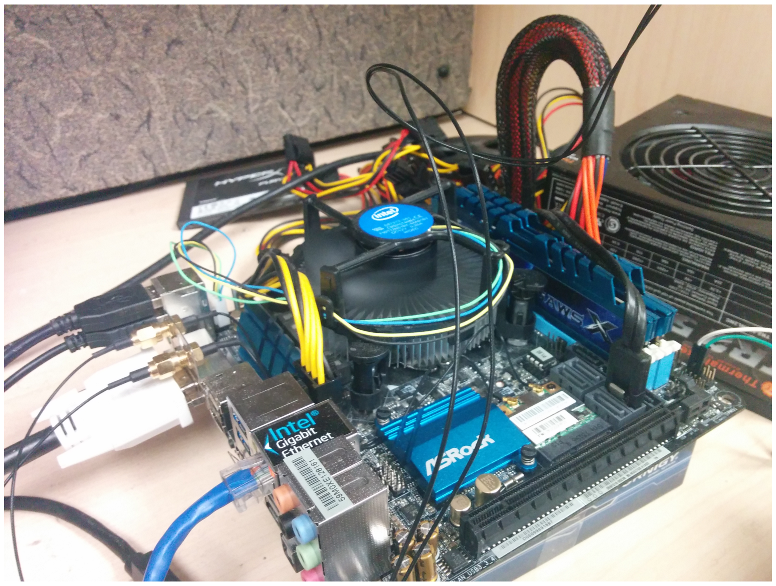


Figure 2.10: Custom Computer for Vision Processing

2.4 Software

The software developed for the WatkeRR is separated into five main categories. The first is the code on the Arduinos that are primarily base level, interacting directly with the hardware formatting and packaging the data for transmission to ROS or Simulink. The second is the code on the computer for integration into ROS. The third is existing and modified ROS packages that perform data collection and control. The fourth is Matlab and Simulink code for data collection and controls development. The final group is image processing and training code.

2.4.1 Arduino Code

The primary function of the software on each Arduino is to read data from the sensors or send commands to the controls. The nature of this structure is pretty flexible but the way it was put together was to put single systems on each Arduino. All of the controller are Uno's except for one that is a mega.

Mega

The mega is used for collecting the data from the Uno's and transmitting it to a processing computer. This is currently accomplished using the built in Bluetooth module. To increase transmission rates the data is muxed on the mega and demuxed on the computer. This is able to run at 20 Hz in real-time in Simulink/Gazebo. The data is read from each Uno via the I2C channel one device at a time. Each Uno is given a unique identifier in there program.

Inertial Measurement Unit (IMU) and ToF Sensors

The IMU and two time of flight (ToF)sensors have I2C connections and the data is routed to the mega for transmission to the system. The IMU has nine DoF accelerometers and magnetometers. The ToF sensors stream a distance measurement of any object in front of them. In this case they are used to find the angle of the user to the WatkeRR.

Encoder Left and Right

The software on each of the encoder controllers reads two encoders (RB-Spa-1042), at 4x encoding it reads 4168 ticks per rotation. The signal from the one encoder is the angle of the wheel, the second is the rotational position of the wheel. This can be then used with the size of the wheel to find the odometry data of the WatkeRR.

Motor Control

This Uno also receives messages through Bluetooth to send Pulse Width Modulation (PWM) signals to the two Electronic Speed Controller (ESC) that control the speed of the rear drive wheels. The other PWM signal is sent to a motor control board that varies the angle of the wheels. A digital signal is required to be sent to the clutch to enable the steering of the wheels.

Wheel Load

The wheel load controller, Wheatstone load amplifier shields (RB-Onl-38), are on each wheel. One amplifier had to have the signal inputs to the Arduino moved using a wire soldered to the board to allow for four sensors to be read simultaneously. The load cell for each wheel is a button load cell (RB-Phi-121) that read linearly from 0-50 kg. The input voltage read from the board is 0-5 voltages that is a digital signal that ranges from 0 to 660 units. The software reads this value and converts it calibrated to newtons force applied to each wheel.

Handle Left, and Handle Right

The handle controllers have the same two Wheatstone load amplifiers shield as the wheel load controller. The loads being read are manually mounted strain gauges attached to the handle discussed in 2.4. The code is calibrated to newtons of force applied down, side to side and grip.

2.4.2 Robot Operating System

To interface with ROS each controller can transmit via Bluetooth, or for consistent data flow a program in python, running on the processing computer, can receive the data using BluePy and convert the data to standard ROS messages. ROS being an open source project has a wide selection of developed software that can be used to increase the functionality of the WatkeRR. Developing the detection software for the ROS environment allows for a standard mechanism that other researchers and robot makers are able to use and build on. The use of ROS also allows for the use of Gazebo. Gazebo is a dynamics based simulation package. It is fully integrated with ROS which allows for quicker development as testing can be done in simulation before it is run on test subjects with the WatkeRR. Packages that we have tested are global_planner for path planning; RTAB for SLAM; YOLO ROS that utilizing the CNN in ROS.

RTAB

The advancement of mapping technologies such as low cost 3D cameras and Light Detection and Ranging (LIDAR) devices have allowed for faster and higher resolution 3D mapping.[20] The ability of the rollators control systems to build a map of its surroundings and know how it is moving in the surroundings is necessary for path planning. By using encoders on the wheels of the WatkeRR and vision based SLAM using the Kinect, the RR is able to track its movements and position accurately in the environment.

2.4.3 Image Based Person Tracking

Global_planner

The Global_planner requires an occupancy map that can either be pre-loaded or built dynamically using a SLAM system. It can then plan a global path using A* or Dijkstra's. This can then be integrated with a local path. The data from the encoders, IMU and the SLAM provide a current pose. The WatkeRR then can control either the wheel angle or the motors to change the path of the WatkeRR. To integrate the object detection, objects of interest will be assessed and be set as a path as the user gets closer to the object.

2.4.4 Gazebo

Gazebo is a simulation tool that is integrated into ROS and can be used to determine the performance of the WatkeRR in software before it is used on real people. The simulation requires a model of the WatkeRR to be loaded. The solid model from solid works was used with moment data for the model to make a SDF file. This file has the model as well as all of the dynamic properties of the model. A basic controller was used to estimate the wheel's angle and power and can be improved with data from the actual WatkeRR.

2.4.5 YOLO/SLAM on ROS

A package was developed using the Darknet framework and thus was able to use trained models to find objects of interest. This package was recently made available. It allows for simple integration of the models that were trained as described later in the thesis.

2.5 Conclusion

The novel design of the RR is discussed in this chapter. The key features are the open source control integration and the adjustable frame. This allows for straight forward testing and validation of multiple sensors and controls projects. The integration of the imaging systems allow for advancement to more advanced machine learning. To move to object detection from the cameras, the images are required to be processed which is discussed in the next chapter.

Chapter 3

Datasets and Image Pre-Processing

3.1 Introduction

Cameras were used on the WatkeRR to collect data for image processing. The initial data was recorded on a Go Pro. A move was made from using the Go Pro to a camera that can be used in a real-time running applications. The main data processing is from the RPi camera. The third set of data used is the ImageNet data-set which is used as a standard. The classification and separation is discussed below.

3.2 Datasets

3.2.1 GoPro Data

We developed a machine-learning system using still frames taken from video from a GoPro camera [13]. The two different setups we used were either, a GoPro camera was worn on the chest of a test subject or mounted on a rollator. The test subject then walked random routes around the university campus and commercial areas. The network was trained for raised objects such as doors, crosswalk buttons, elevator buttons and at the same time features on the ground such as crosswalks, stairs, curbs, and ramps. It also looked for surface materials such as gravel, grass, brick, and snow. This resulted in accuracy of 88.51% for 17 classes over 10 training trials. These results will be used to compare with results from proposed methods[13].

3.2.2 Walker Mounted RPi Data

The data was collected using RPi Camera with a fish eye lens with a 180° field of view. Combining two of the cameras allows for a wide overlapping view in-front and to the sides of the WatkeRR. 270° view around the WatkeRR.

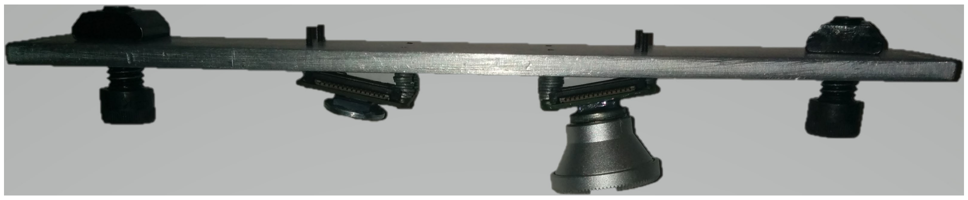


Figure 3.1: Two RPi cameras not mounted on WatkeRR with one of two wide angle lenses

3.2.3 Google ImageNet

ImageNet is a collection of 14.2 million classified images that are curated by google. The database is classified using the WordNet hierarchy. The WordNet 3.0 database is a lexical database of the English language. The words are grouped by concepts that are termed synsets. The ImageNet dataset has images for close to twenty two thousand noun sysnets. As well as the images, there are Scale Invariant Feature Transform (SIFT), bounding boxes, and attributes. As a means to quantify the machine learning algorithms, data scientists held an annual competition called Imagenet Imagenet Large Scale Visual Recognition Challenge (ILSVRC), that has taken place since 2010.. From ILSVRC significant milestones in accuracy and defined network structures have been created.

3.3 Image Classification

One of the most important parts of training neural networks is the data used to train it. This is a task that takes a significant amount of time and effort. For a quality data set, the quantity and accuracy of labels directly affect the result of training. A lot of work was initially done with classifying objects. When working with a network that not only classifies objects but also localizes them, it adds the requirement of precision as well, which can be difficult to control. A custom process was developed to aid data classification and localization. As a means to automatically separate objects the images were converted from

RGB to Hue, Shade, Value/Intensity (HSV). The reason for this is that it was observed that individual objects of interest for this study were found to typically be of similar colour or shade. So the HSV images were automatically segmented and objects were detected using automatic blob detection from OpenCV in each segment.

3.3.1 Image Segmentation

Initially the ability to set the number of segments was variable. This allowed for finding the ideal amount. The benefit of more segments is that there is higher chance of individual objects being found, but it also increases both the chance of similar feature of a single objects being segmented, an increased number of proposals to review, and thus, an increase in processing time. From trials, eight segments per each of the hue, shade, and value, for a total of twenty four segments was found to provide adequate segmentation and speed. The segmentation increased the amount of data to initially be processed by twenty four times but the simplification of the individual arrays, and removal of arrays with minimal data and null arrays provided significantly improved processing times. This can be significantly improved as all processes can be done in parallel using a Graphical Processing Unit (GPU). A TitanXp utilizing the CUDA framework was used for training and testing.

3.3.2 Data Labeling

To label the data collected the segmented blobs were used to speed up the process and allow for accurate bounding boxes. The programs user interface shown in 3.2 would follow the following process:

- load image

    convert RGB image to HSV image

    segment image to 24 parts(3 by 8)

    detect blobs in each of the 24 segments

    sort from biggest to smallest

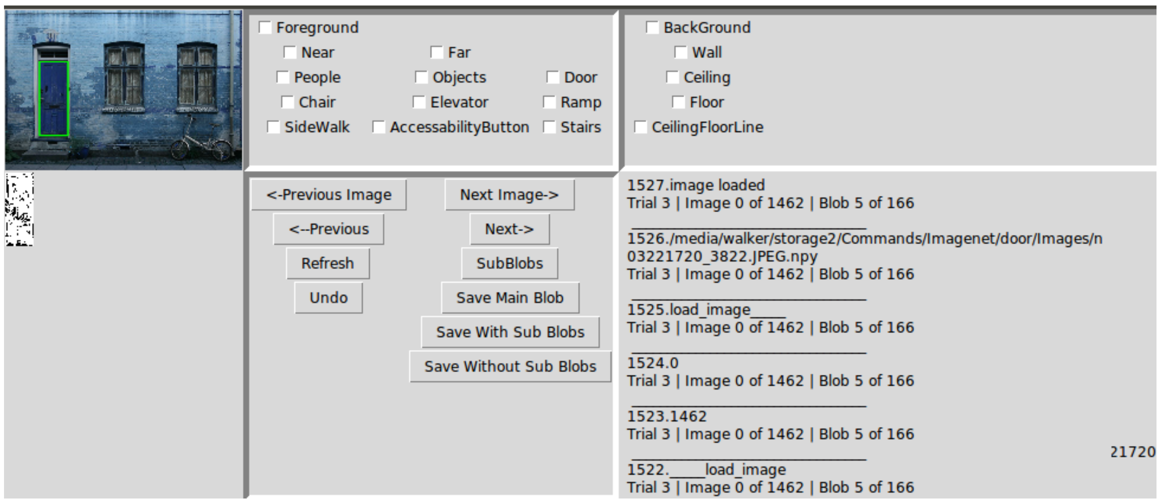
- present blob for classification

    RGB image with a green bounding box drawn in the location of the blob

    contents of the blob

- classification

  blobs and/or blobs located within the bounding box can be classified



```
1527.image loaded
Trial 3 | Image 0 of 1462 | Blob 5 of 166
________________________________
1526./media/walker/storage2/Commands/Imagenet/door/Images/n
03221720_3822.JPEG.npy
Trial 3 | Image 0 of 1462 | Blob 5 of 166
________________________________
1525.load_image____
Trial 3 | Image 0 of 1462 | Blob 5 of 166
________________________________
1524.0
Trial 3 | Image 0 of 1462 | Blob 5 of 166
________________________________
1523.1462
Trial 3 | Image 0 of 1462 | Blob 5 of 166
________________________________
1522.____load_image
Trial 3 | Image 0 of 1462 | Blob 5 of 166
```

Figure 3.2: Display for software of classification and labeling of frames from video data

3.3.3 External Lighting Correction

Indoor and outdoor light is significantly different and the lighting outside can change based on time of day, cloud cover, and weather. This can create challenges for training CNN as similar objects in different lighting can provide very different inputs. To deal with this there are two options; having a training set with examples for each of the conditions, or correcting each image to a standard for the lighting change. The most common method is applying white balancing and normalization to all of the images. The process of white balancing corrects colour to make white in images appear as white during standard day light. This is also known as colour temperature correction. Normalization of the image then would adjust the brightness of the image to an average value. This would make dark images brighter or bright images darker. To measure the effectiveness of white balance and test for alternative methods of brightness correction a test spectrum was made. The spectrum was segmented in eight parts for each hue, shade, and value (HSV). The complete spectrum used for testing is shown in Figure 3.3 The HSV can be seen in 3.4 with the RGB representation seen in 3.5.

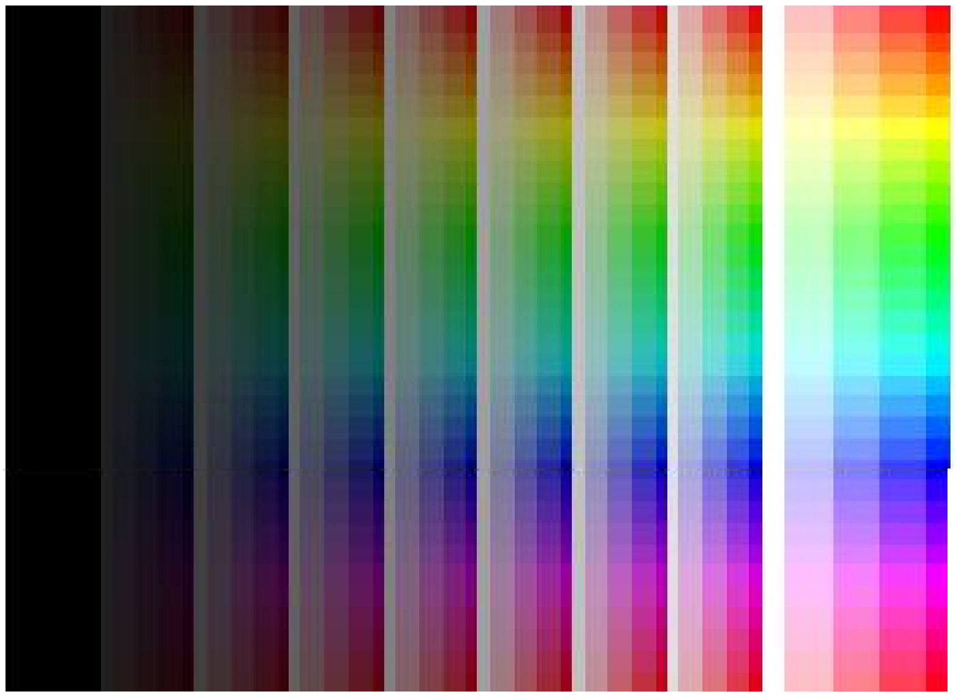


Figure 3.3: Calibration Test Spectrum With HSV

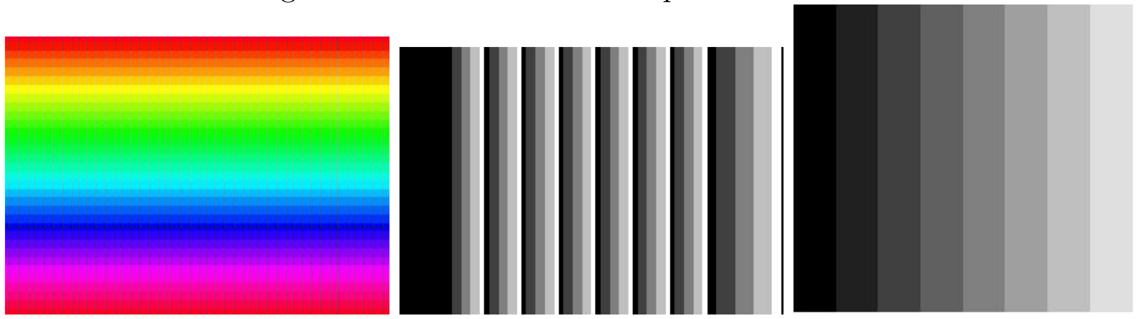


Figure 3.4: Hue, Saturation, Value of Spectrum

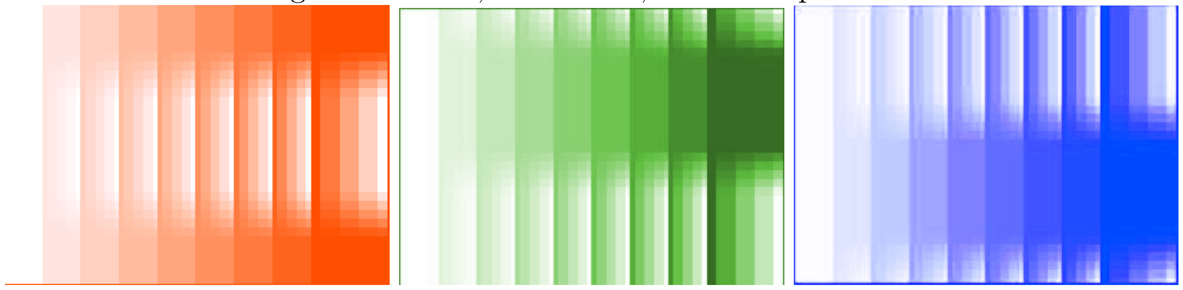


Figure 3.5: Red, Green, Blue Separation of Spectrum

Spectrum were printed and placed in the field of view of the camera, with the intent of measuring the effectiveness of the correction. This would allow for the image to be corrected to a standard HSV independent of the lighting conditions. To collect a training data set, photography equipment was used to project light on the standard spectrum. The equipment used was a large umbrella light to supply diffuse light from different directions, two colour temperature reflectors were used to shift the characteristics of the light. The first step in correcting the image to the spectrum is isolating the part of the image that has the spectrum in it. The simplest method is manually selecting the four corners of the spectrum in the image. This is adequate if the camera and the spectrum are static and always in the same spot. A better method is detecting the spectrum in the image and masking it out. Once the spectrum is isolated the intent is to apply blob separation for HSV. The comparison of the correction will be used to determine the effectiveness of the correction. The correction will then be applied to original images.

3.3.4 Barrel Distortion Correction

The camera has a wide angle lens that increases the area that the camera can see but distorts the image. The image can be corrected using camera calibration features of opencv. To do this a standard 9 inch by 6 inch checkerboard is recorded in multiple different locations and angles. With the spacing of the pattern known, a calibration matrix is obtained. The matrix can be then used to correct the barrel distortion.

3.4 Conclusion

The segmented data was separated into data sets for select parts of the hierarchy proposed in Section 4.4.1. The sets are then duplicated and processed in four categories: unmodified, cropped, blobs, and cropped blobs. The next chapter discusses the use of these data-sets in testing the segmentation. This is done with top layer training re-training of inception CNN. The resulting cross validation accuracy of the loss function is used as a metric for comparing the different data sets.

Chapter 4

Computer Vision and Object Detection

4.1 Introduction

The objective of the machine vision system is to find the objects that a user of the WatkeRR potentially would be interacting with. The focus of this from the bottom up is one object doors, object type, person/object, and near/far. The detected objects can be integrated into the WatkeRR system through ROS. The detected objects will need to be mapped to the map created by the SLAM system. A brief summary of the background of computer vision will be discussed.

4.2 Computer Vision

Computer Vision is a broad topic that covers a variety of techniques used to convert 2D and 3D image and distance data to useful information for automated computer systems. The basics of this field have existed since the late 1960s but have seen an explosion since the 2000s [10]. This is due to the falling cost of processing and constantly improving low cost, high performance, video cards [14]. The applications for computer vision range from image processing, assembly lines, medical detection, object detection, mapping and robotic navigation. All of these applications use an ever growing list of machine learning techniques. From this list, CNNs have been found to be a solution that provides a good results and is very well suited for image processing and computer vision. To get a well trained network that can provide useful automated image processing provides a wide array of challenges:

- Image sets.
  - Large number of images required to train a neural network,tens of thousands to millions in each image set.
  - Standard image sets exist for a wide array of topics. The most popular datasets for object detection are ImageNet, Common Objects in Context (COCO), CIFAR and PASCAL VOC. There is a growing list of specific image sets. The structure and classes for the majority of the data sets is not standard.
- Labels for all the images.
  - The structure and classification for the labels for each dataset is different. The number of classifications can range from two to thousands.
  - ImageNet has 1000s of distinct labels using the Worldnet hierarchy for the structure of the classifications.
- Each trained network is specific to the image set used to train it.
- Structure of the network can have a large effect on performance.
- The number of layers and type of layers need to be determined for each network.
- Image preprocessing is typically performed.
  - The most common modification is cropping and/or resized to make the image smaller. Making the images smaller has the benefit that the model size is reduced.This can significantly reduce the computational processing time of training and operation.
  - Standardizing images can be done in preprocessing such as white balancing.
  - Datasets can be increased by flipping, rotating, shifting, stretching, and shearing.

4.3 Neural Network Structure

The structure for the inception neural network was developed for the ILSVRC 2014 competition, where it was the winner of the annual competition. It was developed by Christian Szegedy working for google. The intent was to reduce the computation required for deep neural networks. The first iteration was called GoogLeNet. The inception v1 is a CNN, that has multiple Network in Network (NiN) layers. The main functional blocks of each layer in the inception model can be seen in Figure 4.1. The main strength of the block is the 1x1 convolutions that significantly reduce the number of dimensions to be processed before the 3x3 and 5x5 layers. The 1x1 convolutions act as a rectified linear units (ReLu) and reduces the number of layers input to the larger convolutions. This is refereed to as bottlenecking as it restricts (bottlenecks) the amount of filters. The initial inception network was basically the structure stacked 9 times in series. The loss function is shown in equation 4.1 as weighted output of all three softmax values.

$$loss_{total} = softmax_{9^{th}} + 0.3 * softmax_{6^{th}} + 0.3 * softmax_{3^{rd}} \tag{4.1}$$

The biggest change to the next iteration of the inception network is that the output of the layers are normalized. This inherently removes biases that need to be corrected if the data was not normalized. The second big change is that the 5x5 and 7x7 filters were removed as they had discovered that stacked 3x3 convolutions could be used to find the same scale of features.

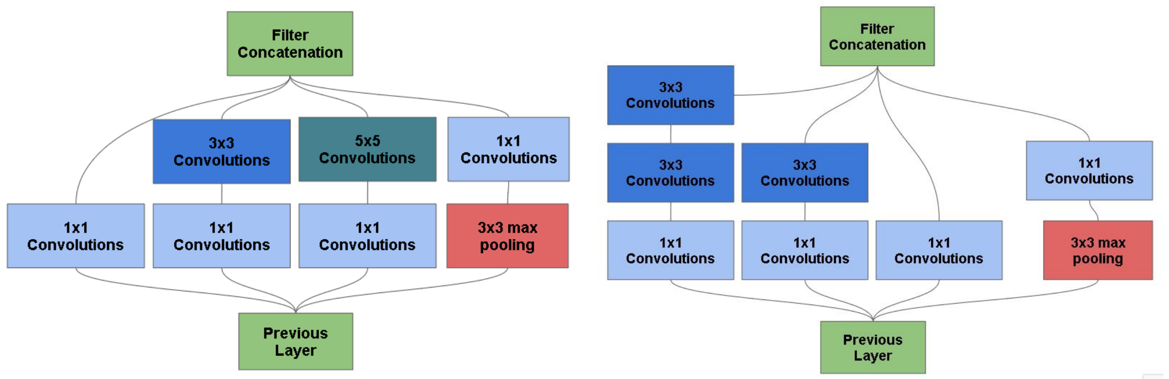


Figure 4.1: Left: Repeated base structure in Inception neural network version 1. Right: Improved repeated base structure in Inception neural network version 2 and 3.

4.3.1 Top Level Re-Training

A significantly faster method for training inception models is top level re-training. With this method the base structure of the model is not retrained. The only part of the model that is trained is the last fully connected layer. This can reduce multi-day training down to 4-5 minutes. The initial weight effectively draws out important image features and the final layer connects the features to the desired classes. If the object trying to be detected has a feature that doesn't fit the existing model, no amount of training will find that feature. Since the inception network is very broad and deep and is trained on 22000 classes initially, this is unlikely.

Comparing the top layer training for 315 separate classes of 279 626 images based on people and objects of interest searched of the synset search versus the same images combined to two classes (people and objects). The separated set gives a test accuracy of 46.3% stopping at 50000 iterations and 97 minutes of training time. The combined data sets gives a cross validated result of 92.7% with 8000 iterations and 10.2 minutes of training. The idea that the fully connected neural network has significantly less neurons and each less neuron requires less weight updates.

4.4 Object Classification

In training neural networks for vision systems, the input is static images or frames from a video and results in classes. A majority of current work is focusing on having a catch all black box that can determine a large number of classes. The down side of this is that training takes significantly longer and accuracy can be lower. The method proposed is shown in Figure 4.2 with a hierarchical structure that allows for training of each layer.

To be able to locate the objects of interest the objects first need to be detected. The first part of the process was training data, using the hierarchy shown in Figure 4.2. The hierarchical structure was designed to break the training data into straight forward logical separation (e.g Near/Far,People/Object ect.). The data was also separated into full RGB 1920x1680 pixel images, 24 spectral separated blobs, and both of these were cropped to the variable size of object class specific bounding boxes. Each of these data sets was trained using Google's Inception v3 top layer training [1]. As part of the top layer training the input images are separated in three groups: training; testing; and validation. From the initial images 10% are separated and used for independent validation after the training. From the remaining 90% of the images each of the training runs 10 fold cross validation using 90% train 10% testing.

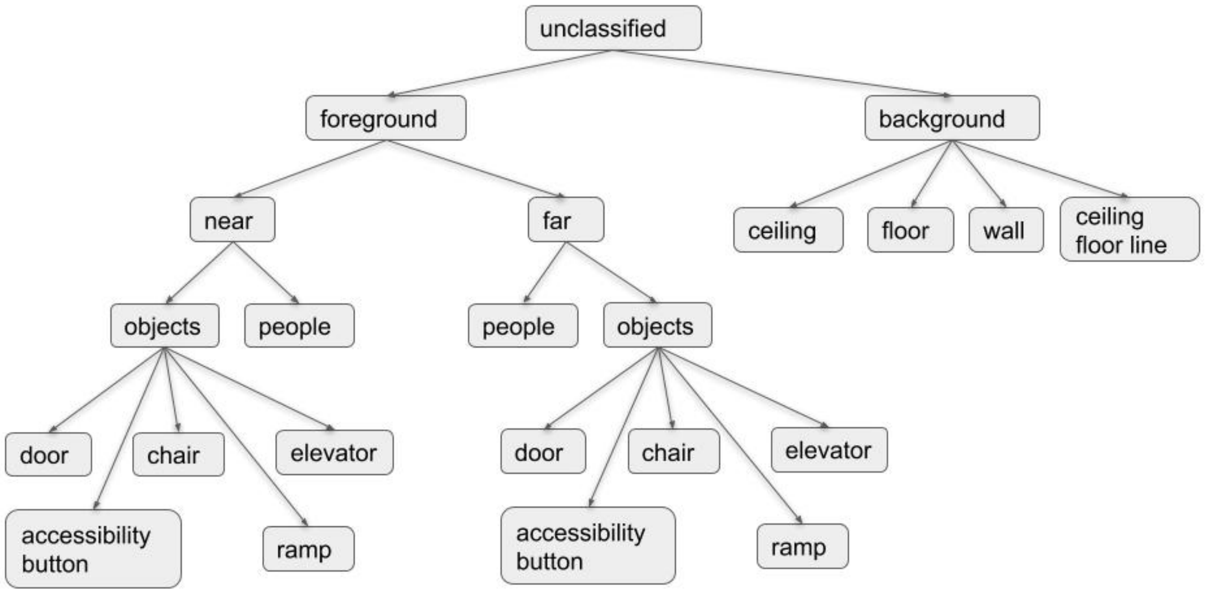


Figure 4.2: Hierarchy of classification for data training of neural networks

4.4.1 ImageNet vs RPi Data

The simplest and most useful metric for comparing the image data sets is the validation accuracy. The final validation accuracy is by running the retrained network on the 10% images not used for training or testing. The training is done for 4000 segments. In testing it was found that a higher number of segments did not significantly improve accuracy. The model was then tested with the segmented set of data and the accuracy found from the test set is the validation accuracy. A comparison of the RPi camera data is shown in Figure 4.3. This is for detecting the object in complete frame images. It can be seen that the accuracy decreases from abstraction down the hierarchy, from 83.3% for separation of foreground and background. The separation of background components into four groups wall, ceiling, floor and separation line of ceiling and floor, results in an accuracy of 54.4%. The validation accuracy for all objects (near and far sets combined) was at 92.4%. The separation of people and the five classes of objects had a logically lower accuracy than directly separating people and objects (74.1%), or directly separating the five groups of objects independently (96.1%). This is found to be slightly higher than the accuracy for separation of the data into a binary door and not door set of 94.5%.

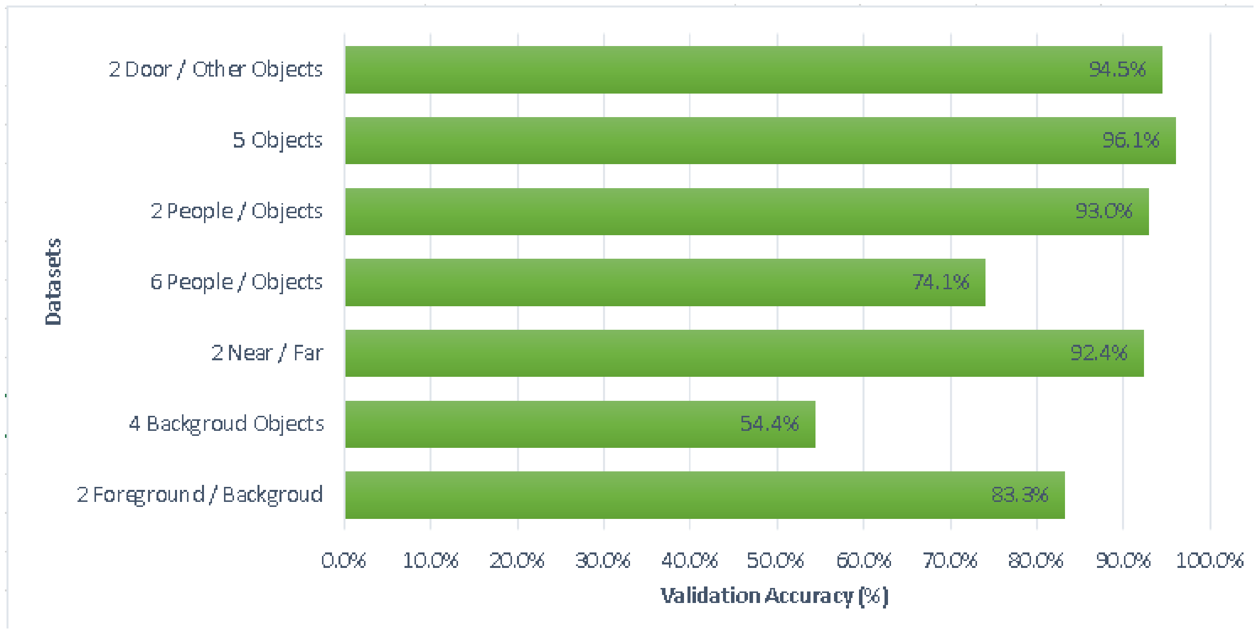


Figure 4.3: Validation Accuracy for Hierarchy of Raspberry Pi Camera Images for 4000 iteration.

To get an objective look at the self acquired and labeled results the same training was performed on the ImageNet dataset. Due to synsets images being limited to nouns, not all categories are available as part of the default set. The sets that are available and a merged set of both sets are compared and shown in Figure 4.4. The three groups of data each have a result that relates to the input data. The first set of two class people and objects, have high and consistent accuracy for R-Pi (93.0%), ImageNet (92.7%) and the two data sets merged (92.6%). The high accuracy and consistency for all sets is expected with two visually distinct classes. Moving to the bottom of the hierarchy to separate object into the five classes; door, elevator, stairs, accessibility buttons, and chairs. The result for the R-Pi 96.1% is on par with the ImageNet result of 96.1%. Even with significantly less data similar results are observed for the RPi data set have limited number of training images. The Merged set for the 5 - Object classes is reduce to 90.1%. This could be a result of differences between the two data sets for the objects.

The final training set merged all the classes, except doors. Doors being the one with the highest number of training images. The merged new class is labeled *NotDoors*. The ImageNet results of 98.2% are the highest seen, which is consistent with the use of a binary

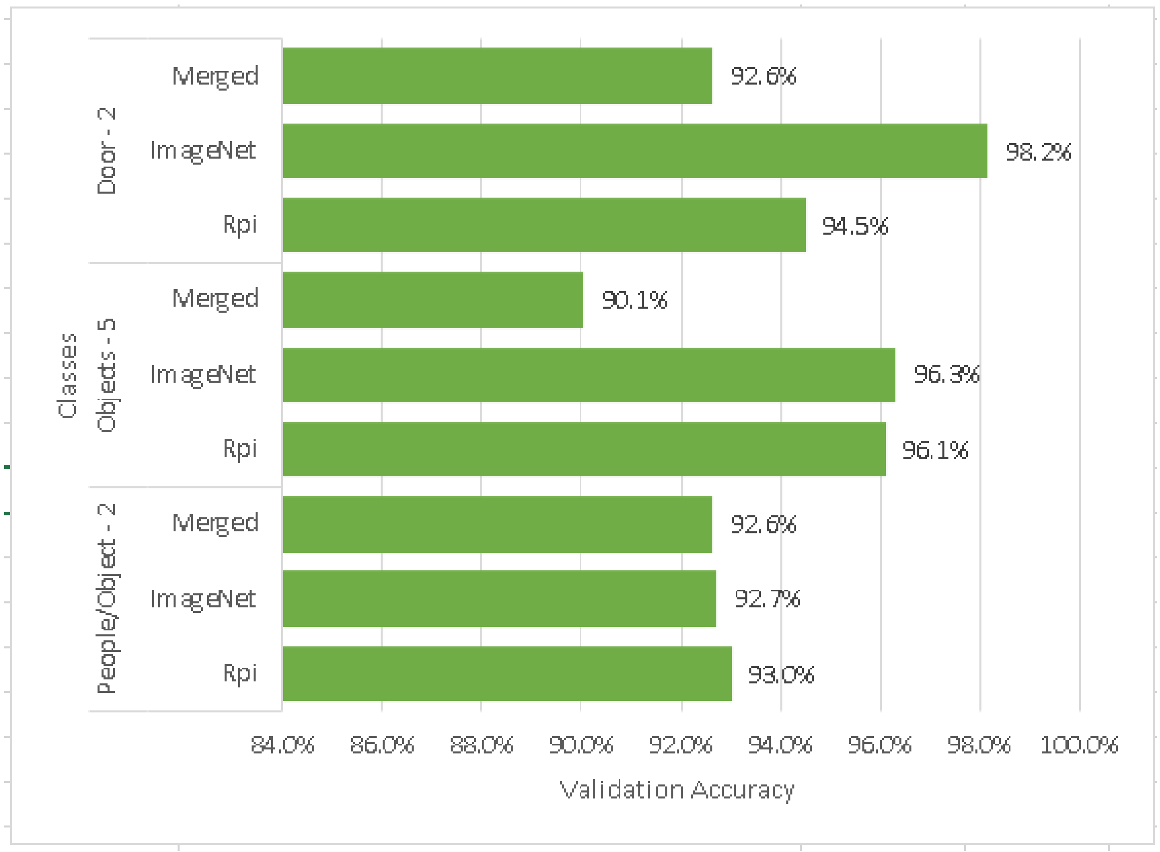


Figure 4.4: Validation Accuracy for Raspberry Pi Camera Images Compared to ImageNet Images and Merged Datasets for 4000 iterations.

class with high quality data. The RPi results is 94.5% which is slightly lower than the 5 class. The merged result is again lower at 92.6%.

4.4.2 Full Images/Cropped Images/Blob Cropped

The results of the top layer retraining of the RPi and ImageNet data sets broken down into hierarchical structure are shown in the Table 4.1. The separation of the standard image, cropped to the object, image with only the HSV blob, and the HSV blob cropped is shown.

Table 4.1: Top Layer Neural Network Training For Broad Image Training

Name	Classes	Validation Acc.(%)	Std. Dev.	Cross Entropy	Std. Dev.	N	Run Time (s)	Std. Dev.
Foreground/Background	2	83.3	2.6	0.307	0.039	786	234.5	4.0
Near/Far	2	92.4	0.7	0.186	0.043	640	233.7	4.5
Near/Far Blob	2	91.8	0.9	0.163	0.037	676	235.2	5.2
Near/Far Cropped	2	92.5	0.9	0.151	0.025	693	238.6	6.9
Near/Far Blob	2	92.2	0.4	0.158	0.033	647	231.9	3.2
People/Object	6	74.1	1.6	0.293	0.038	655	239.0	5.4
People/Object Blob	6	75.6	1.3	0.335	0.048	633	237.2	5.2
People/Object Cropped	6	73.6	1.7	0.265	0.029	686	243.1	4.8
People/Object Blob	6	77.0	1.1	0.299	0.050	711	239.2	5.3
People/Object Imagnet	315	38.1	1.0	3.400	0.050	27911	378.1	12.2
People/Object	5	93.0	0.6	0.177	0.032	698	236.5	8.1
People/Object Blob	5	92.2	0.5	0.155	0.030	684	235.5	6.0
People/Object Cropped	5	90.8	0.7	0.166	0.027	683	246.4	6.6
People/Object Blob	5	91.9	0.7	0.172	0.036	639	237.1	5.5
People/Object Imagnet	2	92.7	0.4	0.159	0.031	28481	246.2	17.0
Object/Object	5	96.1	0.0	0.026	0.016	610	246.3	7.7
Object/Rpi	5	82.5	1.0	0.196	0.032	584	238.2	4.3
Object/Rpi	5	82.5	1.3	0.207	0.037	600	249.1	6.8
Object/Rpi Blob	5	79.2	1.1	0.211	0.033	628	239.2	5.4
Object/Object Imagenet	5	96.3	0.2	0.059	0.019	593	267.7	10.9
Background/Background	4	54.4	1.3	0.525	0.052	161	257.8	9.5
Background/Rpi	4	92.4	0.0	0.100	0.017	119	236.1	5.1
Background/Rpi	4	93.3	0.5	0.108	0.028	140	238.3	6.5
Background/Rpi Blob	4	87.7	0.4	0.096	0.012	147	237.4	4.3
Background/Background	4	92.6	0.2	0.320	0.026	275	252.6	6.7

Table 4.1 continued from previous page

Name	**Classes**	**Validation Acc.(%)**	**Std. Dev.**	**Cross Entropy**	**Std. Dev.**	**N**	**Run Time (s)**	**Std. Dev.**
Door	2	94.5	4.0	0.143	0.086	678	250.2	8.4
Door/Rpi	2	85.6	1.6	0.264	0.046	608	236.6	3.6
Door/Rpi	2	84.2	1.0	0.253	0.036	697	236.5	5.7
Door/Rpi Blob	2	84.8	1.5	0.252	0.039	605	237.7	4.6
Door	2	98.2	0.1	0.022	0.010	568	261.4	103.6
People/Object Imagnet	2	92.6	0.2	0.148	0.031	29065	257.8	14.7
Object/Rpi	2	90.1	0.3	0.198	0.043	1176	244.3	6.9

4.5 Conclusion

The use of top layer training of Inception based CNN has allowed for straight forward assessment of structuring of the training data. The result has demonstrated that the segmented data has an improved accuracy. The results from data collected on the WatkeRR is comparable to the standard set by ImageNet. The door and object results were comparable for the RPi. The next chapter uses the high accuracy data-sets for object detection.

Chapter 5

Object Classification and Detection

5.1 Introduction

Classification of images provides the ability to determine if an object is in a given frame. To be able to find where the classified object is in the image a move to detection needs to be made. The framework used in this paper is Darknet. Darknet is an open source convulutional neural network framework that is developed to detect and create bounding boxes around classified objects. Darknet was originally developed in 2013 by Joseph Redmon [8] as an open source neural network framework. Using Darknet as the feature extraction network, Redmon created YOLO as a Region Proposal Classification Detection Network (RPCDN). Previous RPCDN's such as Recursive Convelutional Neural Network (RCNN) or the performance improved versions fast RCNN and faster RCNN uses multiple recursive passes to form predictions for regions of an image.

Yolo treats the process of image to pixel based bounding box coordinates and classes probabilities as a single regression problem. This uses a single pass that is structured similar to a fully connected CNN. This is done by separating the image into a grid and separating each cell on the grid and creating bounding boxes and confidence of a class for each cell. The confidence value is predicted box compared to the trained ground truth. This single pass method allows for very fast run times on the scale of real time video. The latest release of YOLO is version 3 published in April 2018 that is stated to improve accuracy and run times [16].

5.1.1 YOLO Function

The structure and training of the YOLO network is different from the inception network. The output from the YOLO network is bounding box results versus a found class. To predict the bounding box dimensions, clustering is used. The output of the predictions is four values, the center of the bounding box (t_x, t_y), and the width and height (t_w, y_h).

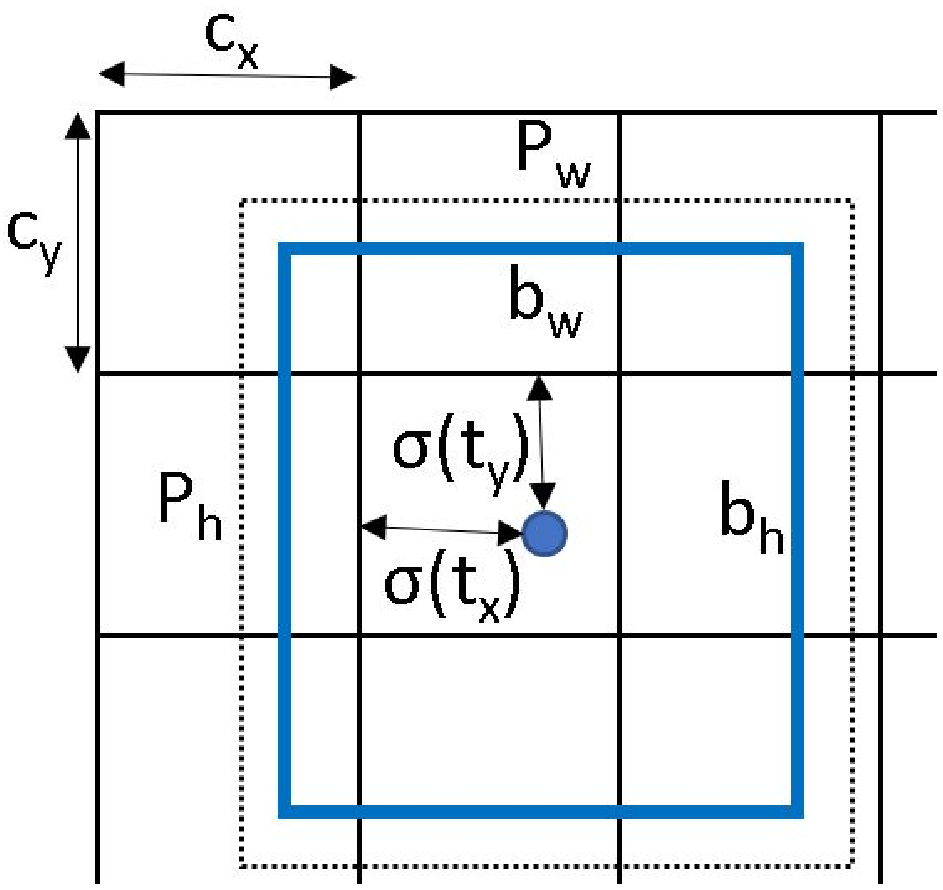


Figure 5.1: Representation of bounding box proposal as part of YOLO v3 [16]

To evaluate the quality of the trained model, we look at the false negatives, false positives, true positive, true positive 50% and 95% for both precision and recall. A true positive 50% or 95% is found if the detected object has over 0.5 or 0.95 of the proposed object overlapped by the ground truth bound box. The precision is the true positive over true and false positives. The recall is the true positive over true positives and false negatives.

5.1.2 Data-sets

For the object, two sets of data were used, the door near objects and the door far objects. The data sets were also combined to make a third data set. The near and far set have 2910 objects and 2421 objects respectively. The combined set has 5331 objects from both sets. The data for each was randomly separated into 90% training and 10% testing sets.

5.2 Results

The precision results shown in figure 5.2 are the three training models combined, far, and near. The three models were then evaluated on the near, far, and combined test and training sets of data. This resulted in 18 sets of data to evaluate for precision and recall. This allows for a comparison of the quality of results and shows the difference of each of the sets. A major difference in re-training the top level vs training the Darknet network is that top level takes from 3-5 minutes where as darknet takes over 1 day for models of this size. This is the result of the Darknet re-weighting the entire graph vs just the top layer training. A confusion matrix is not used here as no true negatives were tested in the validation. What can be found from these results is that the far data model performed

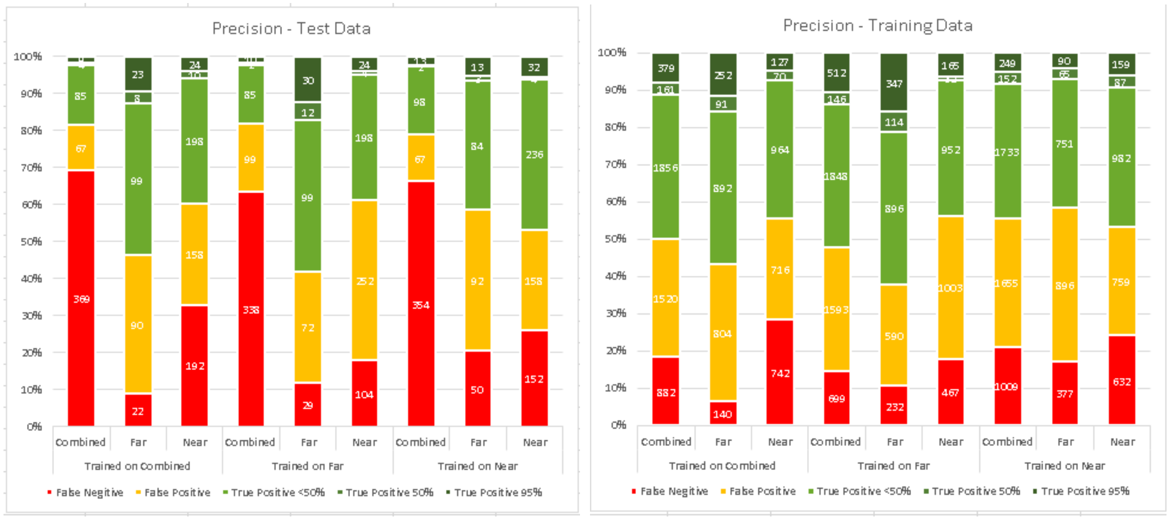


Figure 5.2: Precision on testing and training sets of near, far and combined(near and far) door detection Yolo network on each respective set

the best. On the far test set the results of a positive precision of 58.3% with 17.4% being above 50% and 12.4% being about 95%. Compared to the same model being validated on the training sets, a positive precision of 63.3% with 21.2% above 50% and 15.9% above 95% was found. Similarity between testing and training results is a strong indication that the model is not over trained. The far model also performs well on the combined and near data sets at 53.7% and 41.32% respectively. The near model does not perform as well in testing or training as the far model on the respective data sets. The near model was at 46.74% positive in testing with 6.2% over 50% and of that 5.5% above 95% precision. This is compared to the training results 46.9% with 9.4% over 50% and 6.1% of those above 95%, which is significantly less precise than the far door model. This is opposite of what was expected. The reason for this is likely related to ground truth data errors and will be discussed below. The combined model performed significantly worse in testing than the other two models. The testing precision of 18.4% with a training precision of 49.9% is a strong indication that the model is over-fit.

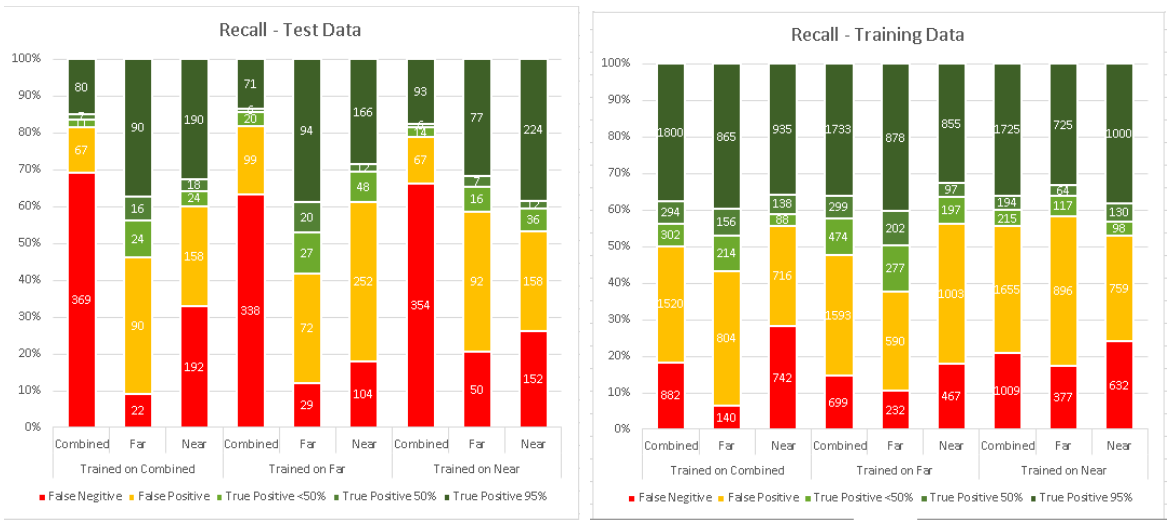


Figure 5.3: Recall on testing and training sets of near, far and combined(near and far) door detection YOLO network on each respective set

The recall shown in 5.3 of the models followed the same trends as the precision with the far model out performing the near and combined models. The far set performed the best on recall as well, with 58.3%. The true positive recall and precision will be the same as a true positive and is independent of found precision or accuracy values. With a 47.1% above 50% and 39% above 95% recall the set trained such that if an object is found, the entire object will be found. With the higher 95% recall compared to precision, it is likely the result of larger than necessary bounding boxes. This will be discussed with the negative examples seen in Figure 5.4 and positive examples seen in Figure 5.5.

Looking at the negative images shown in Figure 5.4 two major issues can be seen. The first that can be seen is that sub features on the door are creating negative results. This could be a result of door signs and or pieces of paper posted on the door being used in the training set. The second major issue is that multiple doors in the images have the effect that a negative result would be recorded since the ground true data is being proposed one at a time. The most straight forward way to avoid this issue is to only use ground truth data that a has a single door in the entire image. This is not ideal as in reality multiple doors will be seen in use, and the YOLO network is designed to find multiple detections in a single image. The other method would be to do an image specific assessment with multiple ground truth at one time.

With these limitations the positive results shown in 5.5 show strength in finding doors in a wide range of distances; open and closed as well as interior and exterior doors. This also appears to not be limited to partial doors. Some issue can be seen with detection of internal features of the doors as well as variation in bounding box sizes.

5.3 Conclusion

The use of the YOLO network for training the detection of doors appears to be very promising. A heuristic filter being applied after detection could easily remove a significant amount of negative values. With the partially autonomous nature of collecting the training data, a further step of classifying sub features of a door may be required.

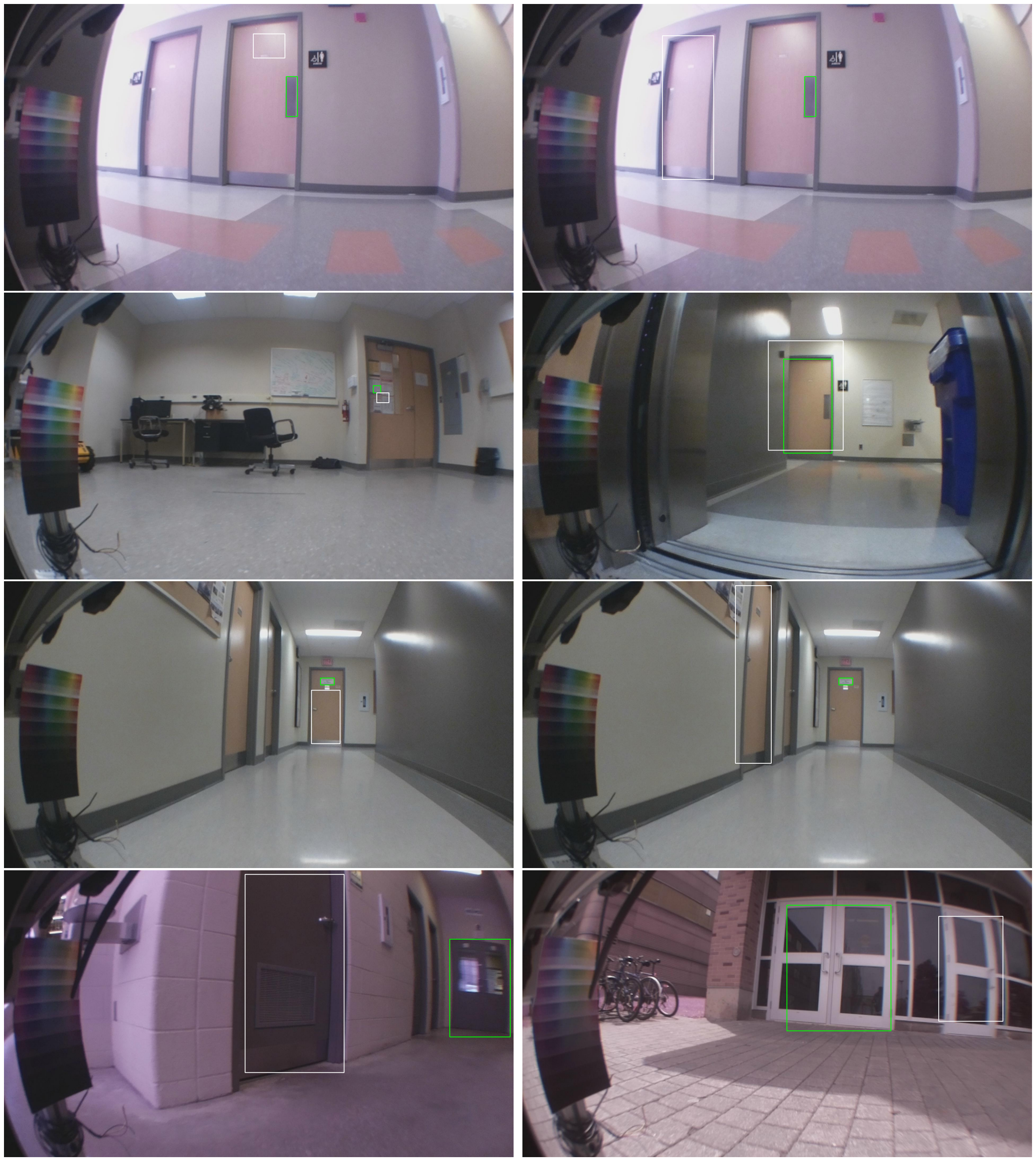


Figure 5.4: Negative Detections in Door YOLO Network. Green boxes indicates ground truth and white boxes are found boxes

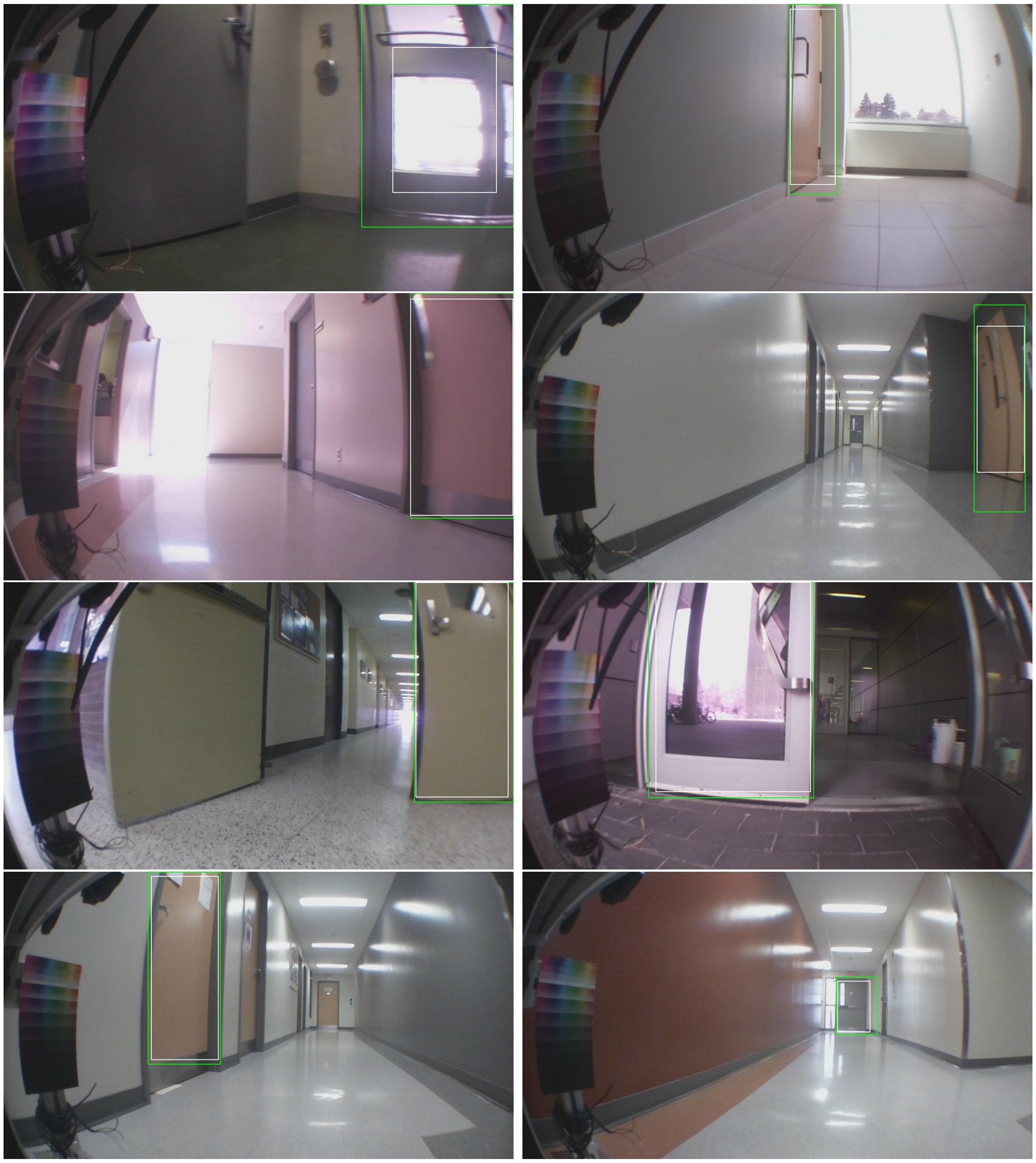


Figure 5.5: Positive Detections in Door YOLO Network. Green boxes indicates ground truth and white boxes are found boxes.

Chapter 6

Conclusions and Future Work

The population of the developed world is moving to a significantly higher proportion of elderly population, posing significant challenges. This challenge will require improvements in efficiency, effectiveness, and cost for elder care. Being able to provide an improved quality of life with a reduced work force will be necessary. The application of low cost robotics have started to assist in this field, but a lot of work needs to be done before they are at a state that can bear the load of the increased demand. The first contribution of this thesis was the development of the WatkeRR. The WatkeRR has allowed for the development of multiple projects that have focused on testing of a wide array of low cost sensors and control algorithms to address the challenging problem of controlling RR as an assist device.

To be able to control the RR, the state of the WatkeRR and user needs to be determined to generate a path that the user would want to travel. The current odometery, IMU, and force sensor data provide the RR state. The time of flight distance sensors and camera are used to be able to get the state of the user. Since the WatkeRR platform was designed to be integrated into the ROS system it can leverage developments made for autonomous robots.

While mapping, object avoidance, and path planning algorithms are all viable, the difference between a RR and an autonomous vehicle is that desired navigation goal for the RR is coming directly from a user that, due to physical or mental limitations, may not be able to input an objective to the system. This creates the challenge of trying to determine the intent of the user. To be able to do more than avoid obstacles, what the user intends to interact with needs to be determined.

A second major research contribution of this thesis is the development and evaluation of a machine visions system to RR. The use of convolutional neural networks has become the common solution to machine vision classification problems such as object detection. The challenge of training convolutional neural networks is that large amounts of training data are required. Improvements in networks trained on standard data sets have given very robust structures for feature extraction. The initial intent was to build a hierarchical structured network. By segmenting data into near/far, people/objects, object types, and specific objects it was found that existing structures such as inception are already well suited for feature extraction.

Using top level retraining on the inception neural network structure allowed for assessment of each of the different input data quickly, such as doors and chairs. Coupled with the use of GPUs to reduce training times from days to minutes, separating each bottom tier hierarchical object into its own network reduced the network complexity and training times. The benefit of the hierarchical structuring and binary separation of the layers being trained allows for all available features found in the CNN can be applied to determining that specific classifier. This can be seen in the resulting higher cross validation accuracy's for the binary classifiers (98.5%). Reducing the same feature extraction to a binary output (desired objects/all other non objects) reduces the training classification further. This benefit is notably significant in top level training of objects that share features in the extraction phase of the network, which may have downstream impact on detection rates. It is possible that with a robust enough top layer that this would occur but the effect of separating the training inputs allows for simpler top level and a definitive separation. This also has the benefit that each classifier can easily be evaluated on its own and retrained independently if needed. Once the input data had been evaluated on the top layer retraining the logic followed using the same input data to train a detection network.

The YOLO network was selected for Darknet because it has the benefit of being used for real time applications, integration in to ROS, as well as ease of integration with data structured for training inception networks. Some structural conversion was required for the bounding box but it was fairly straight forward. Testing only used segmented door features. This was done because it is a more challenging object as it is very visually unique and it is the highest found object in the collected data, and highly relevant in walker navigation and path planning. The training of the YOLO networks on the GPU took a bit over a day as it was a complete retrain. A similar solution as the top level retraining for the inception network would be ideal for training but the structure of Darknet required training of the top two fully connected layers. The results of the validation of the Darknet demonstrated that the combined data sets provided a lower precision and recall than the network trained solely on the far data. It was expected that the near data set would perform better but in reviewing the test results it seems that sub-features of the door (e.g., labels, windows,and posters) caused issues with misclassification and some bounding box errors. The far away doors were less likely to be cut off by the frame and sub features are compressed so the predominant features of the door are more likely to be detected. The continuation of this work is to integrate the detected objects into the path planning of the WatkeRR.

An initial challenge of integrating the detected objects into path planning was communicating the output into ROS, but at the end of 2017 the Darknet ROS package was released that allows for simple integration of ROS camera feeds into object detection as well as detected objects into the ROS network. The next step would be to implement the detection into the WatkeRR, from a detector of an object, such as a door. The dimensions of a standard door can be used to determine the real world coordinates of the door(2040 mm x 926 mm).

With this, trials should be conducted with controlled navigation to the center of found doors, followed by inclusion of other objects, such as chairs, and determining ideal placements and motions associated with interaction of going from a rollator to a chair and vice versa. This could then be done for other key objects influencing walker mobility, such as elevators, ramps, crosswalks and any other situations or objects people may interact with. It would be recommended to expand the data set as it would likely improve accuracy for these other detections. The segmented training would make this straight forward. The next step would be moving from 2D images to 3D data from a depth cameras or LIDAR sensors. This would have the benefit that the real world location would be directly determinable. The difficulty of this is that the data creation for training becomes a lot harder as well as it adds a dimension to the neural network training that can greatly increase training and run times. The addition of the Jetson hardware allows for the removal of the processing computer and significant space savings as well as on board CUDA cores. Integrating the project into a Jetson TX2 would be a worth while project for cost and power reduction. This could streamline the vision system by easily replacing the Raspberry Pi and reduce real time processing times. The TX2 with 6 cameras integrated into it would allow for further options in tracking the user as well.

The nature of the structure of the test frame allows for straight forward collaboration with other researchers. Validation and testing of the proposed controls models as well as calibration and integration of the power steering would directly benefit this project. Unfortunately, due to hardware issues this was not possible to be done at the time of writing. Integrating object classification and detection into assist devices is now possible and necessary for safe and effective operation.

References

[1] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geodrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mande, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow - Large-Scale Machine Learning on Heterogeneous Systems, 2015.

[2] Christine Bizier, Gail Fawcett, and Sabrina Gilbert. Canadian Survey on Disability, 2012 Mobility disabilities among Canadians aged 15 years and older, 2012.

[3] Daniel Castro, Steven Hickson, Vinay Bettadapura, Thomaz Edison, Gregory Abowd, Irfan Essa, and Henrik Christensen. Predicting Daily Activities From Egocentric Images Using Deep Learning. 2015.

[4] Aboriginal Statistics Division. Fact sheet Disability in Canada : Initial findings from the Canadian Survey on Disability, 2013.

[5] L. N Gitlin. Why older people accept or reject assistive technology. *Generations*, 19:41–46, 1995.

[6] Bjarki Hallgrimsson, Jim Budd, and Adrian D.C. Chan. The Smart Rollator Project.

[7] Richard Zhi Ling Hu, Adam Hartfiel, James Tung, Adel Fakih, Jesse Hoey, and Pascal Poupart. 3D Pose tracking of walker users' lower limb with a structured-light camera on a moving platform. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2011.

[8] Joseph Redmon. Darknet: Open Source Neural Networks in C, 2013.

[9] Ulrich Lindemann, Michael Schwenk, Jochen Klenk, Max Kessler, Michael Weyrich, Franziska Kurz, and Clemens Becker. Problems of older persons using a wheeled walker. *Aging Clinical and Experimental Research*, 2016.

[10] Hiroshi Sako Masakazu Ejiri, Takafumi Miyatake, Akio Nagasaka, and Shigeki Nagaya. Evolution of Real-time Image Processing in Practical Application. 2000.

[11] Carlos A Cifuentes Member, Cristina Bayon, Sergio Lerma, Anselmo Frizera Member, and Eduardo Rocon Member. Human-Robot Interaction Strategy for Overground Rehabilitation in Patients with Cerebral Palsy. pages 737–742, 2016.

[12] Caio Benatti Moretti, Ricardo C Joaquim, Thais T Terranova, Linamara R Battistella, Stefano Mazzoleni, and Glauco A P Caurin. Knowledge Discovery strategy over patient performance data towards the extraction of hemiparesis-inherent features : A case study. pages 725–730, 2016.

[13] Mina Nouredanesh, Sunil Kukreja, and James Tung. Detection of compensatory balance responses using wearable electromyography sensors for fall-risk assessment. In *IEEE-EMBC*, Orlando, USA, 2016.

[14] Corporation NVIDIA. GPU-Based Deep Learning Inference : A Performance and Power Analysis. (November):1–12, 2015.

[15] ONU. World population ageing. *United Nations, Department of Economic and Social Affairs, Population Division*, United Nat((ST/ESA/SER.A/390):164, 2015.

[16] Joseph Redmon and Ali Farhadi. YOLOv3: An Incremental Improvement. 2018.

[17] Keith Syrett. Aging, Access, and the Costs of Health Care: Should Canada Query the QALY? *International Journal of Canadian Studies*, 47:41–56, 2013.

[18] Alexander Tettenborn. Obstacle Avoidance in Intelligent Assisted Walking Devices for Improving Mobility Among Seniors with Cognitive and Visual Impairments by. 2016.

[19] James Y. Tung, Justin N. Chee, Karl F. Zabjek, and William E. Mcilroy. Combining ambulatory and laboratory assessment of rollator use for balance and mobility in neurologic rehabilitation in-patients. *Disability and Rehabilitation: Assistive Technology*, 2015.

[20] Christian Werner, Phoebe Ullrich, Milad Geravand, Angelika Peer, and Klaus Hauer. Evaluation Studies of Robotic Rollators by the User Perspective: A Systematic Review. *Gerontology*, 2016.

Appendix A

Code

A.1 Blob Based Data Classifier for WatkeRR - Python

```
# Copyright 2017 WatkeRR Team. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# ==============================================================================

"""
University of Waterloo
Watkerr - www.watkerr.com
Image Classification Program - Take images from a raspberry pi camera array that
have been masked based on sub HSV values then trains the
nerual network to adjust the HSV values to a standard value.

"""
######Package Imports######
import time
import argparse
import os
import stat
import re
import sys
import tarfile
import numpy
import cv2
import tensorflow as tf
from matplotlib.colors import hsv_to_rgb
from matplotlib.colors import rgb_to_hsv
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import matplotlib
import image
import tkinter as tk
from tkinter import filedialog
from PIL import Image
from PIL import ImageTk
from PIL import _tkinter_finder
import cv2
import threading
import pickle


####main graphical user interface#####
class gui():
    def __init__(self,root):
        #Variable Definitions
        self.root=root
        pad=3
        self._geom='200x200+0+0'
        self.root.geometry("{0}x{1}+0+0".format(self.root.winfo_
        screenwidth()-pad, self.root.winfo_screenheight()-pad))
        self.root.title('Blob Classification')
        self.gui_setup(self.root)
        self.main()
        return
```

```
#loads the existing file structure
def filestruct(self):
    self.status('filestruct')
    self.f,self.p=self.filenames(self.root)
    return

##File Managment##
#opens selected directory and returns all the files in it.
def filenames(self,root):
    self.status('filenames')
    cwd = os.getcwd()
    path= tk.filedialog.askdirectory(parent=root,initialdir=
    cwd,title='Please select a directory')
    self.status(path)
    files = []
    subdirs = []
    filenames=[]
    ################################
    st = os.stat(path)
    self.status(str(bool(st.st_mode & stat.S_IRGRP)))
    for rt, dirs, filenames in os.walk(path):
        for subdir in dirs:
            subdirs.append(os.path.relpath(os.path.join(rt, subdir), path))
    for f in filenames:
        files.append(os.path.relpath(os.path.join(rt, f), path))
        self.status(f)
    return filenames,path

##Load##
#Loads numpy array images into program#
def load_image(self,o,filenames,path):
    self.status('_____load_image')
    self.status(str(len(filenames)))
```

```
        self.status(str(o))
        self.status('load_image_____')
        if o in range(len(filenames)):
            pathImage=path+"/Images/"+filenames[o]
            self.status(pathImage)
            image=numpy.load(pathImage)
            self.status('image loaded')
        else:
            self.status('no image loaded')
            image=numpy.zeros(1)
        return image



    #calls the graphical interface.
    def main(self):
        #loads new images when after cycle or out of blobs for that image is cycled
        if self.t_c_image!=self.t_p_image:
            self.t_p_image=self.t_c_image
            self.m_image=self.load_image(self.t_c_image,self.f,self.p)
            self.image_nospec,self.spec=self.spec_crop_simple(self.m_image)
            self.imageSetup()
        self.update_blob()
        if self.t_c_blob>=self.t_t_blob:
            self.status('max blob')
            self.t_c_blob=0
            self.t_c_image+=1
        elif self.t_c_blob<0:
            self.status('min blob')
            self.t_c_blob=self.t_t_blob
            self.t_c_image-=1
        self.imageshow(self.m_image)
        self.blobshow()
        return

        #loads new images when after cycle or out of blobs for that image is cycled
    def imageSetup(self):
            self.status('loading image')
            self.HSVimage,self.specImage=self.load_HSVimage(self.t_c_image,self.f,self.p)
            self.n_image=self.f[self.t_c_image]
        self.image_x,self.image_y,self.image_n=self.HSVimage.shape
            self.image_area=self.image_x*self.image_y
            self.Lmask=self.HSVSplit(self.HSVimage,self.t_t_sect)
        self.blobs_unsorted=[]
        self.blobIntialSetup()
        return

    #Gui Setup
    def gui_setup(self,root):
        self.mc=0
        self.pt1=''
            self.pt2=''
            self.pt3=''
            self.pt4=''
            self.pt5=''
        self.pt6=''
        self.trial=1
            self.t_t_image=0
            self.t_c_image=0
            self.t_p_image=-1
        self.t_t_hsv=3
            self.t_t_sect=8
            self.t_t_blob=0
            self.t_c_blob=0
        self.filestruct()#loads a specified directory.
        self.t_t_image=len(self.f)
```

```
#variable Definition
self.m_state=[]#current state of
self.c_state=0
self.n_image=''
self.m_image=[]
self.m_blobs=[]
self.s_blobs=[]
self.u_blobs=[]
self.m_blob=[]
self.m_blob_image=[]

self.roi=[]
self.cnt=[]
self.saved=0
self.lmaskThresh=5000
self.areaThresh=250
self.saveNumber=0
self.saveList=[]
self.fr=0

self.prev=0

self.ckForeground=tk.IntVar()
self.ckNear=tk.IntVar()
self.ckFar=tk.IntVar()
self.ckPeople=tk.IntVar()
self.ckObject=tk.IntVar()
self.ckDoor=tk.IntVar()
self.ckChair=tk.IntVar()
self.ckTable=tk.IntVar()
self.ckElevator=tk.IntVar()
self.ckBackground=tk.IntVar()
self.ckWall=tk.IntVar()
```

```
        self.ckCeiling=tk.IntVar()
        self.ckFloor=tk.IntVar()
        self.ckCeilingFloorLine=tk.IntVar()
        self.ckElevator=tk.IntVar()
        self.ckRamp=tk.IntVar()
        self.ckSideWalk=tk.IntVar()
        self.ckAccessabilityButton=tk.IntVar()
        self.ckStairs=tk.IntVar()
        self.s_blobs_loaded=0
        self.statusText='Intializing'
#################
        self.classOutput()
        root.configure(background='white')
        w, h = root.winfo_screenwidth(), root.winfo_screenheight()
        root.geometry("%dx%d+0+0" % (w, h))
        self.content = tk.Frame()
        self.status('loading')
        self.buttons = tk.Frame(self.content, borderwidth=5, relief="sunken", width=200, height=100)
        self.checksN = tk.Frame(self.content, borderwidth=5, relief="sunken", width=200, height=100)
        self.checksF = tk.Frame(self.content, borderwidth=5, relief="sunken", width=200, height=100)
        self.button3=tk.Button(self.buttons,text = 'Next Image->', command = self.button3press)
        self.button4=tk.Button(self.buttons,text = '<-Previous Image', command = self.button4press)
        self.button5=tk.Button(self.buttons,text = 'Refresh', command = self.button5press)
        self.button2=tk.Button(self.buttons,text = '<--Previous', command = self.button2press)
        self.button1=tk.Button(self.buttons,text = 'Next->', command = self.button1press)
        self.button6=tk.Button(self.buttons,text = 'Save Main Blob', command = self.button6press)
        self.button8=tk.Button(self.buttons,text = 'Save With Sub Blobs', command = self.button8press)
        self.button9=tk.Button(self.buttons,text = 'Save Without Sub Blobs', command = self.button9press)
        self.button7=tk.Button(self.buttons,text = 'SubBlobs', command = self.button7press)
        self.button10=tk.Button(self.buttons,text = 'Undo', command = self.button10press)
        self.chkForeground=tk.Checkbutton(self.checksN, text='Foreground', variable=self.ckForeground)
        self.chkBackground=tk.Checkbutton(self.checksF, text='BackGround', variable=self.ckBackground)
        self.chkNear = tk.Checkbutton(self.checksN, text='Near', variable=self.ckNear)
self.chkFar = tk.Checkbutton(self.checksN, text='Far', variable=self.ckFar)
self.chkPeopleN = tk.Checkbutton(self.checksN, text='People', variable=self.ckPeople)
self.chkObjectsN = tk.Checkbutton(self.checksN, text='Objects', variable=self.ckObject)
self.chkDoor = tk.Checkbutton(self.checksN, text='Door', variable=self.ckDoor)
self.chkTable = tk.Checkbutton(self.checksN, text='Table', variable=self.ckTable)
self.chkChair = tk.Checkbutton(self.checksN, text='Chair', variable=self.ckChair)
self.chkElevator = tk.Checkbutton(self.checksN, text='Elevator', variable=self.ckElevator)
self.chkRamp = tk.Checkbutton(self.checksN, text='Ramp', variable=self.ckRamp)
self.chkSideWalk = tk.Checkbutton(self.checksN, text='SideWalk', variable=self.ckSideWalk)
self.chkAccessabilityButton = tk.Checkbutton(self.checksN, text='AccessabilityButton', variable=self.ckAccess
self.chkStairs = tk.Checkbutton(self.checksN, text='Stairs', variable=self.ckStairs)
self.chkWall = tk.Checkbutton(self.checksF, text='Wall', variable=self.ckWall)
self.chkCeiling = tk.Checkbutton(self.checksF, text='Ceiling', variable=self.ckCeiling)
self.chkFloor = tk.Checkbutton(self.checksF, text='Floor', variable=self.ckFloor)
self.chkCeilingFloorLine = tk.Checkbutton(self.checksF, text='CeilingFloorLine', variable=self.ckCeilingFloor

self.content.grid(column=0, row=0, sticky=(tk.N, tk.S, tk.E, tk.W))
self.buttons.grid(column=1, row=1, columnspan=1, rowspan=1, sticky=(tk.N, tk.S, tk.E, tk.W))
self.checksN.grid(column=1, row=0, columnspan=1, rowspan=1, sticky=(tk.N, tk.S, tk.E, tk.W))
self.checksF.grid(column=2, row=0, columnspan=1, rowspan=1, sticky=(tk.N, tk.S, tk.E, tk.W))

self.button3.grid(column=1, row=0)
self.button4.grid(column=0, row=0)
self.button2.grid(column=0, row=1)
self.button1.grid(column=1, row=1)
self.button5.grid(column=0, row=2)
self.button7.grid(column=1, row=2)
self.button6.grid(column=1, row=3)
self.button8.grid(column=1, row=4)
self.button9.grid(column=1, row=5)
self.button10.grid(column=0, row=3)

self.chkForeground.grid(column=0, row=0)
```

```
        self.chkNear.grid(column=0, row=2)
        self.chkFar.grid(column=1, row=2)
        self.chkPeopleN.grid(column=0, row=3)
        self.chkObjectsN.grid(column=1, row=3)
        self.chkDoor.grid(column=2, row=3)
        self.chkChair.grid(column=0, row=4)
        self.chkElevator.grid(column=1, row=4)
        self.chkRamp.grid(column=2, row=4)
        self.chkSideWalk.grid(column=0, row=5)
        self.chkAccessabilityButton.grid(column=1, row=5)
        self.chkStairs.grid(column=2, row=5)

        self.chkBackground.grid(column=0, row=0)
        self.chkWall.grid(column=0, row=1)
        self.chkCeiling.grid(column=0, row=2)
        self.chkFloor.grid(column=0, row=3)
        self.chkCeilingFloorLine.grid(column=0, row=4)

        return



    #cut a basic box around the spectrum.
    def spec_crop_simple(self,img):
        self.status('spec crop')
        try:
            print('imageshow')
        except:
            self.status('Error print(img)')

        try:
            print(len(img))
    except:
        self.status('print(len(img))')
    try:
        print(img.shape)
    except:
        self.status('print(img.shape)')
    try:
        imgCrop=img[310:850,50:300]
        img[310:850,0:300]=0
        #lmask=HSVSplit(imgCrop,8)
        self.status('spec crop')
    except:
        self.status('ERROR_CROP')
        imgCrop=[]
    return img,imgCrop

##Crops Blob Images Using a bounding box from the contours##
def blob_crop(self,im,cnt):
    x,y,w,h = cv2.boundingRect(cnt)
    x2=x+w
    y2=y+h
    blob_image=im[y:y2,x:x2]
    return blob_image




def HSVSplit(self,HSVimage,nsect):
    H_range=179
    S_range=255
    V_range=255
    [y,x,hsvLayer]=HSVimage.shape
    shape=(y,x,hsvLayer,nsect)
    Lmask=numpy.zeros(shape)
    for n in range(0,nsect):
        Hrmin=(((n-1)*H_range)/nsect)
        Hrmax=(min(H_range,(n*H_range)/nsect))
        Srmin=(((n-1)*S_range)/nsect)
        Srmax=(min(S_range,(n*S_range)/nsect))
        Vrmin=(((n-1)*V_range)/nsect)
        Vrmax=(min(V_range,(n*V_range)/nsect))
        Lmask[:,:,0,n]=numpy.logical_and(HSVimage[:,:,0]>=Hrmin, HSVimage[:,:,0]<=Hrmax)
        Lmask[:,:,1,n]=numpy.logical_and(HSVimage[:,:,1]>=Srmin, HSVimage[:,:,1]<=Srmax)
        Lmask[:,:,2,n]=numpy.logical_and(HSVimage[:,:,2]>=Vrmin, HSVimage[:,:,2]<=Vrmax)
    Lmask=numpy.array(Lmask)
    return Lmask

##Convert an RGB image to HSV##
def imageRGBtoHSV(self,image):
    imageHSV=cv2.cvtColor(image,cv2.COLOR_RGB2HSV,3)
    return imageHSV

def blobIntialSetup(self):
    self.status('Processing Image Blob Data')
    self.u_blobs=[]
    for hsv in range(self.t_t_hsv):
        for sect in range(self.t_t_sect):
            self.contourarray, self.blobimage,self.mask=self.contour_create(hsv,sect,self.Lmask)
            for b in range(len(self.contourarray)):
                cnt=self.contourarray[b]
                area=cv2.contourArea(cnt)
                if area>self.areaThresh:
                    cntBlobMask=cv2.fillPoly(self.blobimage, cnt, 255)
                    cnt_image = cv2.bitwise_and(self.blobimage, self.blobimage, mask=cntBlobMask)
                    self.blob=blob(hsv,sect, b,'Undefined',cnt,cnt_image,cntBlobMask,area,0,0,0,-1)
                    self.u_blobs.append(self.blob)
        self.m_blobs=self.blob_sort()
        return

    def status(self,Text):
        self.mc+=1
        print(str(self.mc),'--',Text)
        self.pt6=self.pt5
        self.pt5=self.pt4
        self.pt4=self.pt3
        self.pt3=self.pt2
        self.pt2=self.pt1
        self.pt1=str(self.mc)+'.'+Text+'\n'+'Trial '+str(self.trial)+' | Image '+str(self.t_c_image)+'
        of '+str(self.t_t_image)+' | Blob '+str(self.t_c_blob)+' of '+str(self.t_t_blob)
        statusText=self.pt1+'\n _________________________________ \n'+self.pt2+
        '\n _________________________________ \n'+self.pt3+'\n _________________________________ \n'+self.pt4+
        '\n _________________________________ \n'+self.pt5+'\n _________________________________ \n'+self.pt6
        try:
            self.statusPrint = tk.Message(self.content,text=statusText)
            self.statusPrint.grid(column=2, row=1, columnspan=1, rowspan=1, sticky=(tk.N, tk.W))
        except:
            print('gui not intialized yet')
        return

    #updates current person
    def update_blob(self):
        try:
            self.t_t_blob=len(self.m_blobs)
        except:
            self.status('Warning-Blob total did not load')

        try:
            self.m_blob=self.m_blobs[self.t_c_blob]
        if self.m_blob.sn!=-1:
            if self.saved==0:
                if self.prev:
                    self.button2press()
                else:
                    self.button1press()
    except:
        self.status('Warning-Blob total did not load')
        self.m_blob=[]
        self.m_blob_image=[]

    return

#makes the contour.
def contour_create(self,hsl,n,lmask):
    lmask=self.Lmask[:,:,hsl,n]
    LmaskSum=numpy.sum(lmask)
    contours=[]
    im=[]
    hierarchy=[]
    thresh=[]
    threshBinary=[]

    if LmaskSum>self.lmaskThresh:
        im=numpy.multiply(lmask,self.HSVimage[:,:,hsl])
        im=im.astype(numpy.uint8)
        ret,thresh = cv2.threshold(im,1,255,cv2.THRESH_TOZERO)
        ret,threshBinary = cv2.threshold(im,1,255,cv2.THRESH_BINARY_INV)
        [im2, contours, hierarchy] = cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
    return contours,im,threshBinary


def blob_sort(self):
        self.status('Sorting Blobs')
        blobs_sorted=sorted(self.u_blobs,key = lambda blob: blob.area,reverse=True)
        return blobs_sorted

    def sub_blob_find(self,h):
        h_mask=h.cnt_mask
        xh1,yh1,xh2,yh2=h.boundingRectCoords()
        blobs_stack=[]
        for i_index, i in enumerate(self.m_blobs):
            if i.area<(h.area/2):
                xi1,yi1,xi2,yi2=i.boundingRectCoords()
                i_image=i.img
                maskSum=numpy.sum(cv2.bitwise_and(i_image, h_mask))
                if maskSum!=0:
                    if xi1>xh1 and xi2<xh2:
                        if yi1>yh1 and yi2<yh2:
                            blobs_stack.append(i)
        return blobs_stack

    #pause gui
    def pause(self):
        self.pause=1
        return

    #puts main image on graphics
    def imageshow(self,image):

        try:
            self.blobrectangle(image,self.m_blob,255)
            img=cv2.resize(self.m_image, (0,0), fx=0.4, fy=0.4)
            y,x,n=img.shape
            image=Image.fromarray(img,'RGB')
            self.imgTk=ImageTk.PhotoImage(image)
            self.mainImage = tk.Label(self.content,image=self.imgTk,width=x,height=y)
            self.mainImage.grid(row=0,column=0,sticky=(tk.N,tk.W))
        except:
            self.status('ERROR1-ImageShow')
            self.button3press()
        return

     #shows the individual blobs.
    def blobshow(self):
        try:
            c_blob_img=self.blob_crop(self.m_blob.img,self.m_blob.cnt)
            img=cv2.resize(c_blob_img, (0,0), fx=0.4, fy=0.4)
            ret,img = cv2.threshold(img,1,255,cv2.THRESH_BINARY)
            y,x=img.shape
            imgblob=Image.fromarray(img, 'L')
            self.blobTk=ImageTk.PhotoImage(imgblob)
            self.blobImage = tk.Label(self.content,image=self.blobTk,width=x,height=y)
            self.blobImage.grid(row=1,column=0,sticky=(tk.N,tk.W))
        except:
            self.status('ERROR2-BlobShow')
        return

    #puts a rectangle around the blob
    def blobrectangle(self,image,blob,clr):
        try:
            self.x,self.y,self.w,self.h = cv2.boundingRect(blob.cnt)
            cv2.rectangle(image,(self.x,self.y),(self.x+self.w,self.y+self.h),(0,clr,0),2)
        except:
            self.status('No blob center found')
        return

    def blobSave(self,setting):
        self.status('Saving')
    try:
        state=[]
        #Save Main Blob
        if setting==0:
            self.m_blob=self.NNSave(self.m_blob,'m')
         #Save With Sub Blobs
        elif setting==1:
            self.m_blob=self.NNSave(self.m_blob,'mb')
            self.loadSubBlobs()
            for sb_index,sb in enumerate(self.s_blobs):
                snPrename='sb'+str(self.m_blob.sn)
                self.s_blobs[sb_index]=self.NNSave(sb,snPrename)
        #Save Without Sub Blobs
        elif setting==2:
            self.m_blob=self.NNSave(self.m_blob,'m')
        else:
            self.status('ERROR3-save mode not set')
        state=[self.saveNumber,setting,self.m_blob,self.s_blobs,self.t_c_blob,self.t_c_image]
        self.m_state.append(state)
        self.c_state+=1
        self.deChkBox()
        self.saved=1
    except:
        self.status('ERROR4-blob save')
    return


def NNSave(self,blob,preName):
    try:
        self.saveNumber+=1
        blob.sn=self.saveNumber
        blob.categories=[[self.ckForeground.get(),self.ckBackground.get()],[self.ckNear.get(),
        self.ckFar.get()], [self.ckPeople.get(), self.ckObject.get()],[self.ckDoor.get(), self.ckChair.get(),
        self.ckTable.get(),self.ckElevator.get(),self.ckRamp.get(),self.ckSideWalk.get()
        ,self.ckAccessabilityButton.get(),self.ckStairs.get()], [self.ckWall.get(),self.ckCeiling.get(),s
        elf.ckFloor.get(),self.ckCeilingFloorLine.get()]]
        blob.name=str(self.saveNumber)+preName+'_hsv'+str(blob.hsv)+'_sect'+str(blob.sect)+'_b'+
        str(blob.b)+'_n'+str(self.n_image)+'_c'+str(blob.categories)
        blobSaveName=self.pathTrial+str(blob.name)
        blobSaveName=blobSaveName.replace('npy','p')
        pickle.dump(blob, open(blobSaveName,'wb'))
        self.saveList.append(blobSaveName)
    except:
        self.status('ERROR5-NNSave')
    return blob

def blobUndo(self):
    self.status('Undo')
    try:
        state=self.m_state[-1]
        self.m_state.pop()
    except:
        self.status('Cannot Undo')
    try:
        self.t_c_blob=state[4]
        self.t_c_image=state[5]
        self.m_blob=state[2]
        self.m_blob.sn=-1
        self.s_blobs=state[3]
        if state[0]==self.saveNumber:
            os.remove(self.saveList[-1])
            self.saveList.pop()
        else:
            for a in range(state[0],self.saveNumber):
                os.remove(self.saveList[-1])
                self.saveList.pop()
            self.saveNumber=state[0]
        except:
            self.status('ERROR6-Undo fault')
        return

    def classOutput(self):
        self.pathSorted=self.p+'/sorted/'
        if not os.path.exists(self.pathSorted):
            os.makedirs(self.pathSorted)

        while self.fr!=1:
            self.pathTrial=str(self.pathSorted)+'/'+str(self.trial)+'/'
            if os.path.exists(self.pathTrial):
                self.trial+=1
            else:
                os.makedirs(self.pathTrial)
                self.fr=1
        return

    def loadSubBlobs(self):
        self.m_image=self.load_image(self.t_c_image,self.f,self.p)
        self.s_blobs=self.sub_blob_find(self.m_blob)
        for sb in self.s_blobs:
            self.blobrectangle(self.m_image,sb,125)
        self.s_blobs_loaded=1
        return

    def load_HSVimage(self,o,filenames,path):
        self.status('load_HSVimage')
        pathHSVImage=path+'/HSV/'
        self.status(pathHSVImage)
        if not os.path.exists(pathHSVImage):
            os.makedirs(pathHSVImage)
            self.status('HSV directory made')

        if o in range(len(filenames)):
            self.status('set paths')
            pathRGBImage=path+'/Images/'+filenames[o]
            pathHSVImage=path+'/HSV/'+filenames[o]
        try:
            self.status(pathHSVImage)
            HSVimage=numpy.load(pathHSVImage)
            self.status('HSV Image Load Successful')
        except:
            self.status('Image failed to load')
            try:
                self.status('import HSV image')
                HSVimage=self.imageRGBtoHSV(numpy.load(pathRGBImage))
                numpy.save(pathHSVImage,HSVimage)
            except:
                self.status('ERROR No Image Found')
                HSVimage=numpy.array([])
        try:
            self.status('Try crop')
            HSVimage,specImage=self.spec_crop_simple(HSVimage)
        except:
            self.status('ERROR - HSV -Image load')
            specImage=[]
        return HSVimage,specImage


    def setChkBox(self):
        save=1
        s1=self.ckDoor.get()+self.ckChair.get()+self.ckTable.get()+self.ckElevator.get()
        +self.ckRamp.get()+self.ckSideWalk.get()+self.ckAccessabilityButton.get()+self.ckStairs.get()
        if s1!=0:
            self.ckObject.set(1)
        s2=self.ckObject.get()+self.ckNear.get()+self.ckFar.get()+self.ckPeople.get()
        if s2!=0:
            self.ckForeground.set(1)
        s3=self.ckWall.get()+self.ckCeiling.get()+self.ckFloor.get()+self.ckCeilingFloorLine.get()
        if s3!=0:
            self.ckBackground.set(1)

        if self.ckForeground.get():
            s4=self.ckNear.get()+self.ckFar.get()
            if s4==0:
                self.status('To save please select near or far')
                save=0
        s5=s1+s2+s3
        if s5==0:
            self.status('To save please select a classification')
            save=0
        return save

    def deChkBox(self):
        self.ckForeground.set(0)
        self.ckBackground.set(0)
        self.ckNear.set(0)
        self.ckFar.set(0)
        self.ckPeople.set(0)
        self.ckObject.set(0)
        self.ckDoor.set(0)
        self.ckChair.set(0)
        self.ckTable.set(0)
        self.ckElevator.set(0)
        self.ckWall.set(0)
        self.ckCeiling.set(0)
        self.ckFloor.set(0)
        self.ckCeilingFloorLine.set(0)
        self.ckElevator.set(0)
        self.ckRamp.set(0)
        self.ckSideWalk.set(0)
        self.ckAccessabilityButton.set(0)
        self.ckStairs.set(0)
        return

    #Functions for the buttons.
    #Next Blob
    def button1press(self):
        self.status('next')
        self.prev=0
        self.t_c_blob+=1
        self.s_blobs_loaded=0
        #print('1',self.t_c_image,' ',self.t_c_blob)
        self.saved=0
        self.button5press()
        return
    #previous Blob
    def button2press(self):
        self.status('prev')
        self.prev=1
        self.t_c_blob-=1
        self.s_blobs_loaded=0
        #print('2',self.t_c_image,' ',self.t_c_blob)
        self.saved=0
        self.button5press()
        return
    #next image
    def button3press(self):
        self.status('Next Image')
        self.t_c_image+=1
        self.t_c_blob=0
        self.s_blobs_loaded=0
        #print('3',self.t_c_image,' ',self.t_c_blob)
        self.deChkBox()
        self.status('Next Image...Loaded')
        self.root.update()
        self.main()
        return
    #previous Image
    def button4press(self):
        self.status('Previous Image')
        self.t_c_image-=1
        self.t_c_blob=0
        self.s_blobs_loaded=0
        #print('4',self.t_c_image,' ',self.t_c_blob)
        self.deChkBox()
        self.status('Previous Image...Loaded')
        self.root.update()
        self.main()
        return
    #refresh image
    def button5press(self):
        #self.status('Refreshed')
        self.s_blobs_loaded=0
        self.m_image=self.load_image(self.t_c_image,self.f,self.p)
        self.imageshow(self.m_image)
        self.root.update()
        self.main()
        return
    #Save Main Blob
    def button6press(self):
        save=self.setChkBox()
        if save:
        self.status('Saving')
        self.s_blobs_loaded=0
        self.blobSave(0)
        self.status('Saving...done')
        self.button1press()
        return

    #Save With Sub Blobs
    def button8press(self):
        save=self.setChkBox()
        if save:
        self.status('Saving')
        self.blobSave(1)
        self.status('Saving...done')
        self.button1press()
        return
    #Save Without Sub Blobs
    def button9press(self):
        save=self.setChkBox()
        if save:
        self.status('Saving')
        self.s_blobs_loaded=0
        self.blobSave(2)
        self.status('Saving...done')
        self.button1press()
        return
    #Undo
    def button10press(self):
        self.status('Undo')
        self.s_blobs_loaded=0
        self.blobUndo()
        self.status('Undo...done')
        self.button5press()
        return

    #sub blobs
    def button7press(self):
        self.status('Loading sub blobs...')
        self.loadSubBlobs()
        self.status('Loading sub blobs...Done')
        self.root.update()
        self.main()
        return

    def callback(self,event):
        self.roi.append([event.x,event.y])
        print(self.roi)
        return


##########################################################
##### Save state ######
class blob:
    def __init__(self,hsv,sect,b,name,cnt,img,cnt_mask,area,level,parent,categories,sn):
        self.hsv = hsv
        self.sect = sect
        self.b = b
        self.name = name
        self.cnt = cnt
        self.img = img
        self.cnt_mask = cnt_mask
        self.area = area
        self.level = level
        self.parent = parent
        self.categories = categories
        self.sn=sn
    def moments(self):
        self.moments=cv2.moments(self.cnt)
        return self.moments
    def area(self):
        self.area = cv2.contourArea(self.cnt)
        return self.area
    def perimeter(self):
        self.perimeter = cv2.arcLength(self.cnt,True)
        return self.perimeter
    def epsilon(self):
        self.epsilon = 0.1*cv2.arcLength(self.cnt,True)
        self.approx = cv2.approxPolyDP(self.cnt,self.epsilon,True)
        return self.epsilon,self.approx
    def convexHull(self):
        self.convexHull = cv2.isContourConvex(self.cnt)
        if self.convexHull:
            self.hull = cv2.convexHull(self.cnt,returnPoints = False)
            self.defects = cv2.convexityDefects(self.cnt,self.hull)
        else:
            self.hull=[]
            self.defects=[]
        return self.convexHull,self.hull,self.defects
    def boundingRectCoords(self):
        self.x1,self.y1,self.w,self.h = cv2.boundingRect(self.cnt)
        self.x2=self.x1+self.w
        self.y2=self.y1+self.h
        return self.x1,self.y1,self.x2,self.y2
    def boundingRectRect(self):
        self.rect = cv2.minAreaRect(self.cnt)
        return self.rect
    def bouningBox(self):
        self.box = numpy.int0(cv2.boxPoints(self.rect))
        return self.box
    def boundingCircle(self):
        (self.cx,self.cy),self.radius = cv2.minEnclosingCircle(self.cnt)
        return self.cx,celf.cy,self.radius
    def boundingEllipse(self):
        self.ellipse = cv2.fitEllipse(self.cnt)
        return self.ellipse
    def crop(self):
        self.boundingRectCoords()
        cropImage=self.img[self.y1:self.y2,self.x1:self.x2]
        return cropImage



###########Main Code##################
if __name__ == '__main__':
    root=tk.Tk()
    x=gui(root)
    x.root.mainloop()
```

A.2 Top Layer Bulk Training Script

```
#!/bin/bash
mkdir /media/walker/storage2/Results/D1/
bazel-bin/tensorflow/examples/image_retraining/retrain --image_dir /media/walker/storage2/Datasets/D1_imagenet_people_object
--summaries_dir /media/walker/storage2/Results/D1/summary/door_people --output_graph /media/walker/storage2/Results/D1/Graphs
--output_labels /media/walker/storage2/Results/D1/Labels/ --print_misclassified_test_images &> /media/walker/storage2/Results/D1/outputlog.txt
mkdir /media/walker/storage2/Results/D2/
bazel-bin/tensorflow/examples/image_retraining/retrain --image_dir /media/walker/storage2/Datasets/D2_people_object
--summaries_dir /media/walker/storage2/Results/D2/summary/door_people
--output_graph /media/walker/storage2/Results/D2/Graphs --output_labels /media/walker/storage2/Results/D2/Labels/
--print_misclassified_test_images &> /media/walker/storage2/Results/D2/outputlog.txt
mkdir /media/walker/storage2/Results/D3/
bazel-bin/tensorflow/examples/image_retraining/retrain --image_dir /media/walker/storage2/Datasets/D3_WalkerDataCombined
--summaries_dir /media/walker/storage2/Results/D3/summary/door_people --output_graph /media/walker/storage2/Results/D3/Graphs
--output_labels /media/walker/storage2/Results/D3/Labels/ --print_misclassified_test_images &> /media/walker/storage2/Results/D3/outputlog.txt
mkdir /media/walker/storage2/Results/D4/
bazel-bin/tensorflow/examples/image_retraining/retrain --image_dir /media/walker/storage2/Datasets/D4_WalkerDataPeopleandObjectsCombined
--summaries_dir /media/walker/storage2/Results/D4/summary/door_people
--output_graph /media/walker/storage2/Results/D4/Graphs --output_labels /media/walker/storage2/Results/D4/Labels/
--print_misclassified_test_images &> /media/walker/storage2/Results/D4/outputlog.txt
mkdir /media/walker/storage2/Results/D5/
bazel-bin/tensorflow/examples/image_retraining/retrain --image_dir /media/walker/storage2/Datasets/D5_WalkerDataPeopleandObjectsCombined_CROPED
--summaries_dir /media/walker/storage2/Results/D5/summary/door_people
--output_graph /media/walker/storage2/Results/D5/Graphs
--output_labels /media/walker/storage2/Results/D5/Labels/
--print_misclassified_test_images &> /media/walker/storage2/Results/D5/outputlog.txt
mkdir /media/walker/storage2/Results/D6/
bazel-bin/tensorflow/examples/image_retraining/retrain --image_dir /media/walker/storage2/Datasets/D6_door_chair
--summaries_dir /media/walker/storage2/Results/D6/summary/door_people
--output_graph /media/walker/storage2/Results/D6/Graphs --output_labels /media/walker/storage2/Results/D6/Labels/
--print_misclassified_test_images &> /media/walker/storage2/Results/D6/outputlog.txt
mkdir /media/walker/storage2/Results/D7/
bazel-bin/tensorflow/examples/image_retraining/retrain --image_dir /media/walker/storage2/Datasets/D7_door_elevator
--summaries_dir /media/walker/storage2/Results/D7/summary/door_people
--output_graph /media/walker/storage2/Results/D7/Graphs --output_labels /media/walker/storage2/Results/D7/Labels/
--print_misclassified_test_images &> /media/walker/storage2/Results/D7/outputlog.txt
mkdir /media/walker/storage2/Results/D8/
bazel-bin/tensorflow/examples/image_retraining/retrain --image_dir /media/walker/storage2/Datasets/D8_door_people
--summaries_dir /media/walker/storage2/Results/D8/summary/door_people -
-output_graph /media/walker/storage2/Results/D8/Graphs --output_labels /media/walker/storage2/Results/D8/Labels/
--print_misclassified_test_images &> /media/walker/storage2/Results/D8/outputlog.txt
mkdir /media/walker/storage2/Results/D9/
bazel-bin/tensorflow/examples/image_retraining/retrain --image_dir /media/walker/storage2/Datasets/D9_door_ramp
--summaries_dir /media/walker/storage2/Results/D9/summary/door_people
--output_graph /media/walker/storage2/Results/D9/Graphs --output_labels /media/walker/storage2/Results/D9/Labels/
--print_misclassified_test_images &> /media/walker/storage2/Results/D9/outputlog.txt
```

A.3 Darknet Train Test File Creation - Python

```
import glob, os

# Current directory
current_dir = '/media/walker/storage2/Datasets/DS15_Object_Rpi_Yolo_Near/data/'
# Directory where the data will reside, relative to 'darknet.exe'
data_dir = '/media/walker/storage2/Datasets/DS15_Object_Rpi_Yolo_Near/cfg/'

# Percentage of images to be used for the test set
percentage_test = 10;

# Create and/or truncate train.txt and test.txt
file_train_path=str(data_dir)+'train.list'
file_test_path=str(data_dir)+'test.list'

file_train = open(str(file_train_path), 'w')
file_test = open(str(file_test_path), 'w')

# Populate train.txt and test.txt
counter = 1
index_test = round(100 / percentage_test)
for pathAndFilename in glob.iglob(os.path.join(current_dir, "*.jpg")):
    title, ext = os.path.splitext(os.path.basename(pathAndFilename))
    if counter == index_test:
        counter = 1
        file_test.write(current_dir + title + '.jpg' + "\n")
    else:
        file_train.write(current_dir + title + '.jpg' + "\n")
        counter = counter + 1

file_test.close()
file_train.close()
```

A.4 Darknet Validation

```
#!/bin/bash
#load file locations
DATASETDIR1=/media/walker/storage2/Datasets/DS15_Object_Rpi_Yolo/cfg/
DARKNETDIR=/media/walker/storage/Andrew/Workspace/darknet
SAVEFILE15=/media/walker/storage2/Results/DS15_Object_Rpi_Yolo_Near/MAPOUTPUT_test_16.txt
SAVEFILE16=/media/walker/storage2/Results/DS16_Object_Rpi_Yolo_Far/MAPOUTPUT_test_16.txt
SAVEFILE19=/media/walker/storage2/Results/DS19_Object_Rpi_NearFar/MAPOUTPUT_test_16.txt
#SAVEFILEBB=/media/walker/storage2/Results/DS15_Object_Rpi_Yolo_Near/BB_test_15.txt
SAVEFILEBB=/media/walker/storage2/Results/DS16_Object_Rpi_Yolo_Far/BB_test_16.txt
#SAVEFILEBB=/media/walker/storage2/Results/DS19_Object_Rpi_NearFar/BB_test_19.txt

#list of images to test
#TESTFILE="/media/walker/storage2/Datasets/DS15_Object_Rpi_Yolo_Near/cfg/test.list"
TESTFILE="/media/walker/storage2/Datasets/DS16_Object_Rpi_Yolo_Far/cfg/test.list"
#TESTFILE="/media/walker/storage2/Datasets/DS19_Object_Rpi_NearFar/cfg/test.list"

COUNT=0
cd $DATASETDIR1
ls
#process all the images in TESTFILE
echo "Reading File" $TESTFILE
FIRSTLINE=""
while read -r
do
    COUNT=$((COUNT+1))
    line="$REPLY"
    echo "Image - $line"
    cd $DARKNETDIR
    #Reads the existing bounding box data.
    BOUNDINGBOXFILE="${line/".jpg"/".txt"}"
    read -r -a FIRSTLINE < "$BOUNDINGBOXFILE"
    echo "___" ${COUNT} ${FIRSTLINE[1]} ${FIRSTLINE[2]} ${FIRSTLINE[3]}
    ${FIRSTLINE[4]} ${FIRSTLINE[5]} >>$SAVEFILEBB
    echo "___" ${COUNT} >> $SAVEFILE15
    echo "___" ${COUNT} >> $SAVEFILE16
    echo "___" ${COUNT} >> $SAVEFILE19
    #validate each model seperatly models.
    /media/walker/storage/Andrew/Workspace/darknet/darknet detect
    /media/walker/storage/Andrew/Workspace/darknet/cfg/yolo-voc.2.0_amm_r0_1_testing.cfg
    /media/walker/storage2/Results/DS15_Object_Rpi_Yolo_Near/yolo-voc_final.weights "$line"
    -thresh 0.05>>$SAVEFILE15||echo "FAIL 15">>$SAVEFILE15

    /media/walker/storage/Andrew/Workspace/darknet/darknet detect
    /media/walker/storage/Andrew/Workspace/darknet/cfg/yolo-voc.2.0_amm_r0_1_testing.cfg
    /media/walker/storage2/Results/DS16_Object_Rpi_Yolo_Far/yolo-voc_final.weights "$line"
    -thresh 0.05>>$SAVEFILE16||echo "FAIL 16">>$SAVEFILE16

    /media/walker/storage/Andrew/Workspace/darknet/darknet detect
    /media/walker/storage/Andrew/Workspace/darknet/cfg/yolo-voc.2.0_amm_r0_1_testing.cfg
    /media/walker/storage2/Results/DS19_Object_Rpi_NearFar/yolo-voc_final.weights "$line"
    -thresh 0.05>>$SAVEFILE19||echo "FAIL 19">>$SAVEFILE19
#write to file
    echo ${COUNT}"---"  >> $SAVEFILE15
    echo ${COUNT}"---"  >> $SAVEFILE16
    echo ${COUNT}"---"  >> $SAVEFILE19

    cd $DATASETDIR1
done < "$TESTFILE"
```