Survivor Identification and Retrieval Robot

By

Zhijie Jin

Karun Koppula

Zachary Wasserman

Final Report for ECE 445, Senior Design, Spring 2018

TA: Xinrui Zhu

9 May 2018

Project No. 15

Abstract

A differential drive robotic platform is proposed to implement and test real-world applicability of cutting edge exploration algorithms. A fully ROS integrated system with a forward mounted manipulator that can be used for exploration and object retrieval. Single Shot Detection is used to classify and localize objects in the workspace and to direct retrieval protocols. Object retrieval testing was mostly successful with individual components working as desired, but design errors prevented full performance.

Contents

Introduction

Purpose

After the Fukushima Daiichi disaster caused by the 2011 tsunami, robotics proved largely incapable of responding to the disaster. Massive amounts of funding have since been allocated to solving the problems that prevent robots from operating in unconstrained and unknown environments. Many teams developed different platforms which have since been used in disaster response situations [1]. DARPA, in particular, has a Robotics challenge that required robots to perform tasks that might be required in a disaster scenario including interacting with objects, maneuvering in uneven and unknown terrain, and even driving a vehicle. Unfortunately progress in the these directions have not made significant enough progress to be able to reliably use in disaster scenarios [2].

Different disaster response robots are currently being used, however, as documented and run by the Center for Robotic-Assisted Search and Rescue. These robots are for the most part human operated and are limited by human restrictions and the ability to communicate with the operator. Since these robots are not being used for heavy manipulation, their primary task is surveilance in inaccesible areas.

In disaster scenarios, it is often the case where finding survivors is a time critical operation given their possible medical states. Having an autonomous robotic platform that is onsite and can immediately begin exploration of the environment would increase the overall response time dramatically. It would reduce the time that it takes for first responders to arrive on the scene, set up their robotics systems, and establish safety parameters for operating in the environment. If an autonomous system can do a search and survey and present accurate information to the first responders, they can act immediately and potentially safe lives.

In recent years, work has begun on using machine and reinforcement learning techniques to develop robotic systems that can navigate through a simulated environment using visual and distance data, that eliminates the need for classical navigation tasks such as localization and trajectory planning.

At the 2017 NIPS conference Pieter Abeel gave a keynote presentation about the use of meta-learning which is an algorithm that can learn a policy for a reinforcement learning task, referencing work done by Mnih [3] This method greatly cuts down on the number of training episodes needed to converge to an optimal behavior for each new environment. It allows the system to generate a general policy for exploration that doesnt overfit to a particular environment. This presentation inspired this project, but is beyond the necessary requirements for this project.

We attempted to develop a real world platform that can function as a test bed for exploration algorithms as both a tool to learn about the challenges involved with implementing these algorithms on a real robot and as a precursor for more advanced and capable robots.

Functionality

The robot has been designed to perform object detection and retrieval of a goal object. It is fully integrated and developed with the Robot Operating System, or ROS, and leverages many ROS functionalities. It uses differential driving capabilities to move in the plane and Single Shot Detection to classify and localize objects in its field of view. Once the goal object is detected, it performs retrieval by moving to the object and grasping it with the attached manipulator. The robot then returns to a ending position.

Subsystem Overview

The physical breakdown of the robots components are shown in the block diagram given by Figure 1. The blocks are grouped in a standard fashion for robots. The entire robot is controlled by a microprocessor, a Raspberry Pi which runs all of the lower level robot controls as well as all of the algorithmic processing. A high-level overview of the software is presented in Figure 2. The robot also contains a camera and an array of ultrasonic sensors that feed input about the world to various components of the software. They are grouped into the sensor block. The actuators are seperated into the driving motors and the manipulator servos.

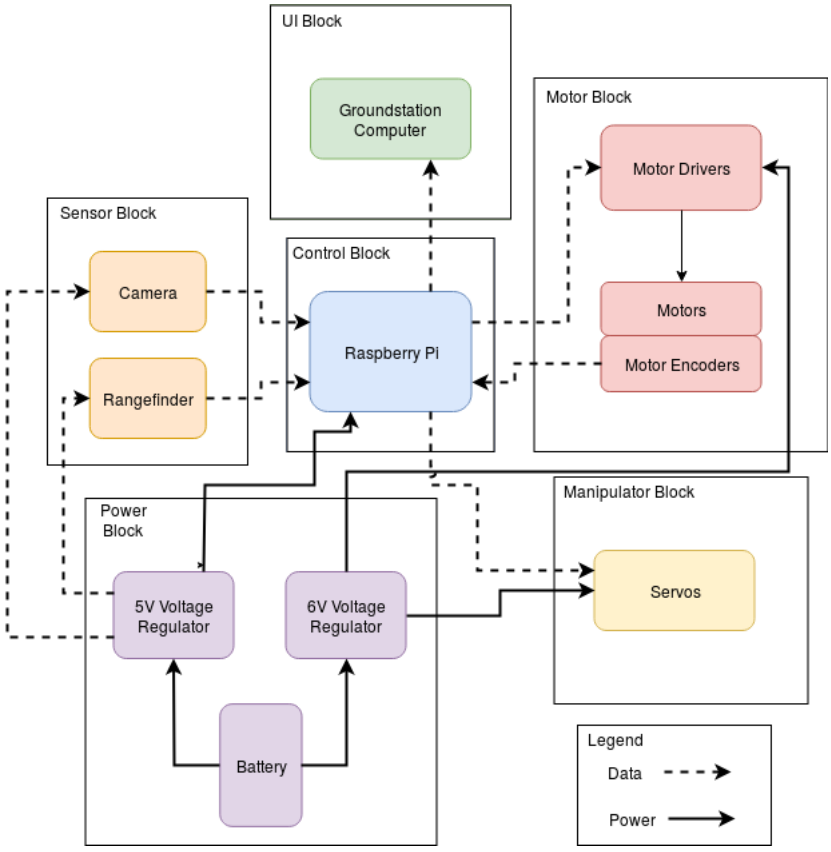

Figure 1: Block Diagram

Nearly the entirety of the project was done in software, however, so the high-level overview is presented to show how the different sections of functionality connect across both hardware and software. The robot takes goal inputs from either the exploration block or the object detection block, based on the identification of

the goal object and the start of the retrieval routine. It passes that goal to the ROS navigation stack, which generates a trajectory to the goal pose. The corresponding linear and angular velocities needed to realize this trajectory are passed to the differential drive controller, which dictates the speed at which the motors should turn. The sensors pass data in at multiple points to provide the different blocks with information about the environment surrounding the robot.

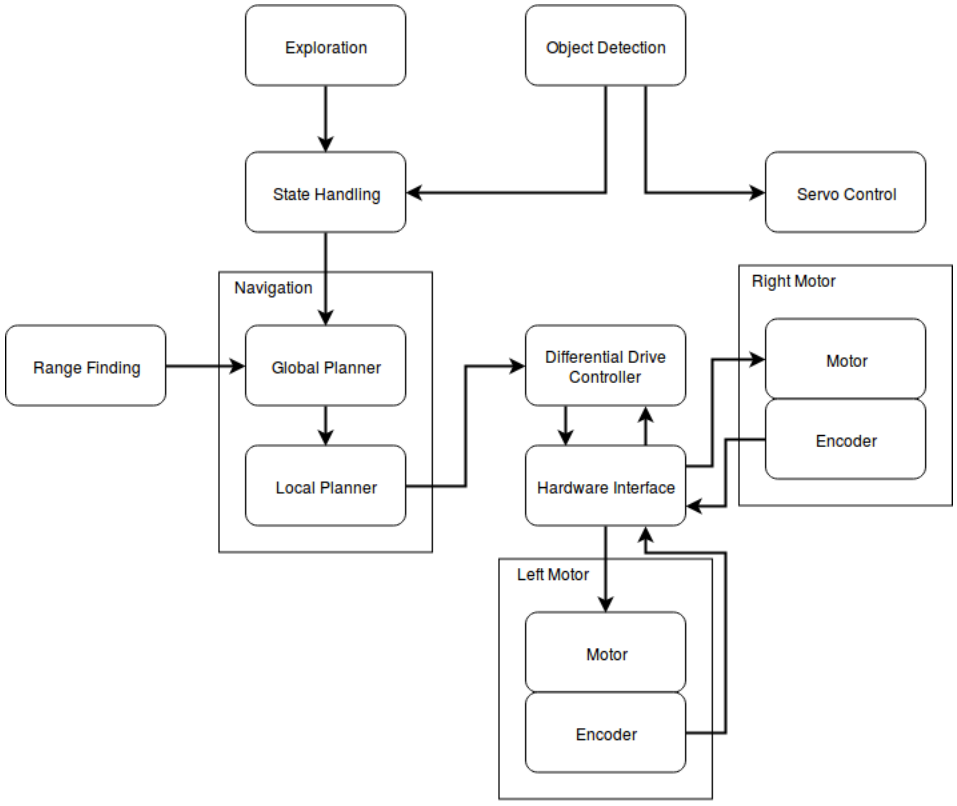

Figure 2: Software Diagram

Differential Navigation

Differential Driving

Design

The differential drive capability is the core component of this robot. We chose to use brused DC gearmotors at the recommendation of the machine shop. Brushless DC and stepper motors are also often used for robots, but for accurate velocity control with a PWM, pulse-width modulation, signal we decided that the motors we chose would work the best. The control loop is shown in Figure 3.

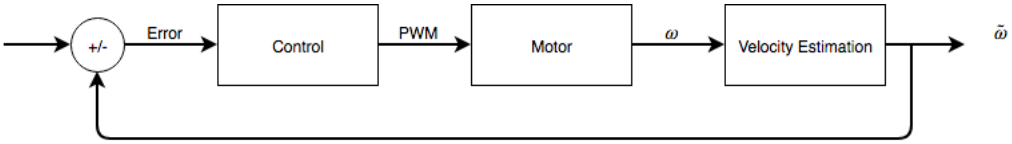

Figure 3: Wheel Control Loop

We used a PID, proportional-integral-derivative, control loop to control the velocities of both wheels. The gains were manually tuned with a randomly seeded gradient descent approach. ROS provides a dynamic reconfigure functionality that allows the user to manipulate values within the code as it runs, which proved instrumental in tuning the controller. In order to create the control loop, we need the current velocity of the wheel. The chosen motors come with an integrated quadrature encoder that allows us to determine the position and velocity at which the wheels are spinning. We created a encoder driver to read the position and velocity of the wheels, implementing and infinite horizon filter on the calculated velocity measurement in an attempt to reduce the noise. Unfortunately, because the Raspberry Pi does not have a realtime operating system, the driver missed some rotations of the wheel when the signal frequency was at about 2 kHz, which is at the upper end of the wheel capability. The output wheel velocity value is still an estimation of the actual velocity, which adds nonlinearities to the control system.

We used a ROS differential drive controller to implement the differential drive equations. The differential drive equations are given by Equations 1 and 2.

$$v = \frac{r}{2}(\omega_r + \omega_l) \tag{1}$$

$$\omega = \frac{r}{L}(\omega_r - \omega_l) \tag{2}$$

The differential drive controller solves these equations for the individual wheel velocities, where r is the radius of the wheel and L is the seperation between the wheels. It takes as input a linear velocity, v and an angular velocity, ω from the navigation stack that corresponds to the desired trajectory.

Verification

The verification of the differential driving starts with the performance of the velocity controller for the wheels. The step response of the controller is shown in Figure 4. The response is within the performance

requirements that were set. The overshoot is less than ten percent and the rise time is close to one tenth of a second. The settling time, however is longer than expected. During the course of tuning the controller gains, it became apparent that the derivative gain had little effect on the response, possibly because of the signal noise. Since the sign of the derivative is changing almost continuously, the collective derivative control effort cancels out. Tuning the proportional and integral gains became an issue of managing the oscillations. The final controller gains are given below.

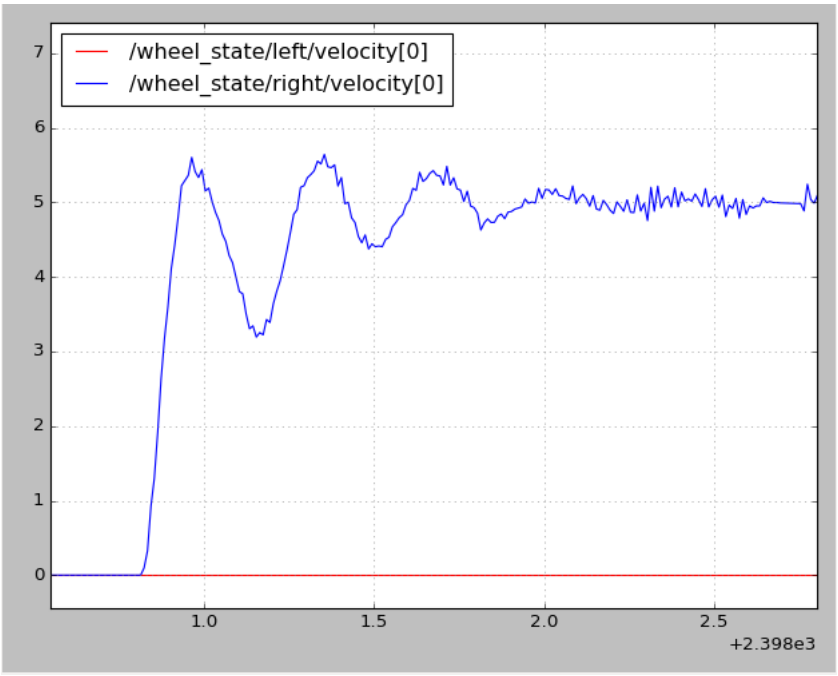

Figure 4: Step Response

$$k_p = 35.0 \tag{3}$$

$$k_d = 620.0 \tag{4}$$

$$k_i = 110.0 \tag{5}$$

After verifying the controller's ability to track an input velocity, we tested the performance across the range of feasible wheel velocities, up to $11\frac{rad}{s}$. The results of that testing is given in Appendix A. Three important discoveries were made during this experiment. First, it showed that for velocities below $3\frac{rad}{s}$, the controller would hit the lower saturation limit. This limit occurs because the control signal does not provide enough voltage across the motor to overcome is inertia and does not produce movement. In order to average the desired velocity, the controller produced an sinusoidal response as shown in Figure 5.

The second discovery was that because of the non-realtime nature of the Raspberry Pi, it would schedule other tasks ahead of the PWM output to the motor controllers and there would be no output to the motors. This happened more often above $7\frac{rad}{s}$. We confimed that it was the Raspberry Pi and not an issue with the power circuity by probing the PWM output from the Pi as the controller was running. An example of this

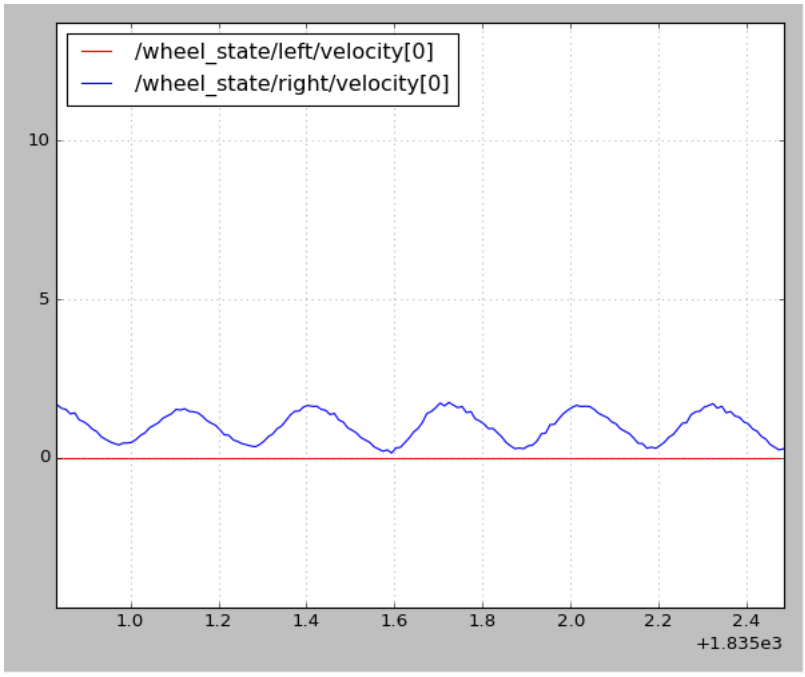

Figure 5: Slow Response

is shown by Figure 6. The third thing of note was the proportionally growing noise in the velocity signal with increased reference velocities.

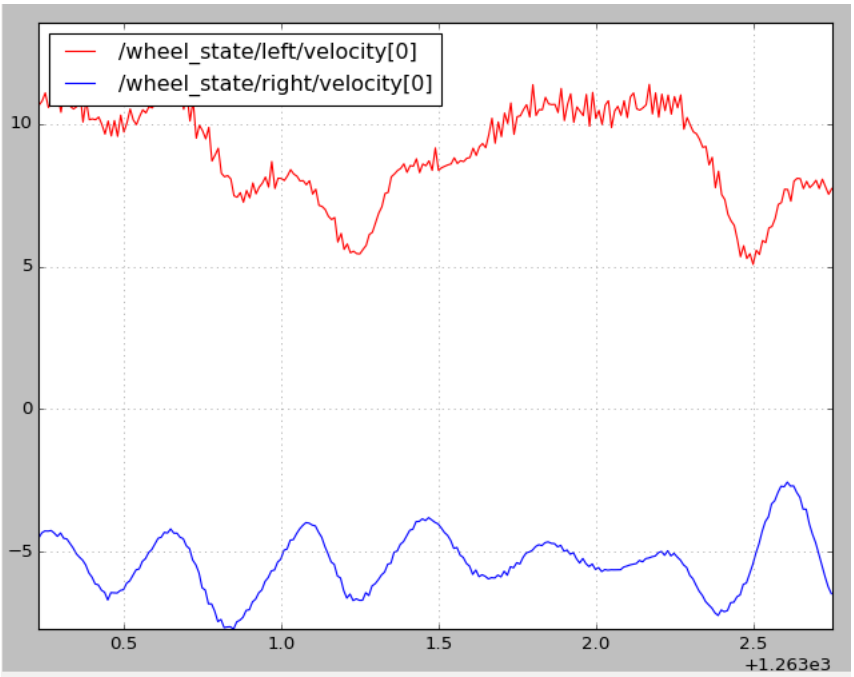

Figure 6: Lost Control Effort

The last verification for the differential driving system was to check the performance of the wheels to given linear and angular velocities. The results are shown in Appendix A. Cells marked with an "X" are where

the wheel displayed either an oscillatory reponse or none at all. The sample points were spread out across the feasible range for the robot.

Unfortunately because of the pace of out circuitry development, we were not able to run these tests with the robot actually moving around. All the verification was done with the robot mounted on a block and the wheels free-running. We were not able to account for the inertial effects of the frame of the robot on the performance of the actual driving behavior. While we showed that the driving system was accurate within tolerance, we were not able to perform slight motions in actual movement that we could do while free-running.

Navigation

Design

We utilized the ROS navigation stack to control the motion of the robot. The navigation stack takes in a goal position and guides the robot along a trajectory to that position. It creates a global plan using the carrot planner, which simply takes the straight line trajectory to the nearest unoccupied position to the goal position. We used this planner because it was the least computationally complex, and would allow us to approach the goal object instead of trying to avoid it. The local planner uses the Dynamic Window Approach to track the global trajectory. DWA samples the control space and simulates applying the control to see what the best control output is to move towards the goal and stick to the global plan. It outputs the angular and linear velocities to the differential drive controller to move the robot. The planners use a costmap to place and avoid obstacles. The costmaps are generally populated by a laser scanner device, but we used a fake laser scanner made up of our ultrasonic sensors. Since this pseudo-device had poor angular resolution and high noise, it would populate the costmap with obstacles even when there were none. We were unable to use the pseudo-scanner in the navigation, which added the additional complication of not having a fixed object position in the costmap. This hampered the robot's ability to move accurately to the goal object position.

Verification

We verified the navigation system by setting goal poses in the odometric frame and checking that the final position was within a tolerance of the desired value. Through the local planner we were able to set this goal tolerance and show that the robot moved correctly. The actions that were tested were a forward movement of 0.3 meters, and right and left turns each by 90 degrees. Even with successive motions, the robot moved to within the tolerance value. During the object retrieval, however, the robot needed to make much smaller rotations and the required trajectories would output velocities below the threshold to overcome to moment of inertia and the robot would get stuck. The planners can only move on to a new action at the successful completion or abortion of the previous action, so the robot would not be able to make the appropriate turns.

Exploration

Design

We were unable to implement the reinforcement algorithm originally proposed due to issues setting up the training environment and working with the reinforcement learning coding libraries. We chose to instead implement and use the A* algorithm used for finding the shortest path through a maze environment. We know that the path returned by the algorithm is the optimal one in the maze because our heuristic is both admissable and consistent. We used the Manhattan distance between the current and goal cells as the heuristic, which is the distance along grid lines between the two cells. The way A* algorithm works is by using a heuristic to be able to decide what the next move at a given point should be. A heuristic is a hint for the maze navigator, this can be implemented in many different ways.

Verification

Because of the issues faced with the navigation software, we were unable to test the actual robot performance using the commands given by the search, but we were able to show that the actions required to navigate to the goal could be output to the navigation stack sequentially. We also were able to verify that the calculated path was indeed optimal by comparing it to the shortest path output by a breadth-first search of the maze.

Object Detection and Localization

Single Shot Detection

Design

The implementation of the classification (what object is in a picture) and the localization (where is the object in the picture) are handled by the Single Shot Detection algorithm. The steps for the algorithm are as follows.

1. Pass image through convolutional layers to give feature maps.
2. For each location use convolutional filter to evaluate default bounding boxes.
3. For each box predict bounding box offset and class probabilities.
4. During training match ground truth with predicted boxes.

What this means is we first try to find the important features of an object in the frame. For humans this would be the head/feet/arms/torso or any other part that the algorithm decides is important. The presence of these are weighted when deciding the probability of what an object is. During the whole process of the algorithm, it is going to be iterating over thousands of boxes within the image. Hence, for part two at each box we will iterate over each of these boxes to evaluate them. What we are evaluating is the loss. That is, how far is the box from being accurately drawn and is the classification correct. This is modeled by equation 6. The loss for a particular bounding box is given by a linear combination of the loss due to the error in the confidence of the objects classification and the loss incurred due to the location error of the bounding box.

$$multiBoxLoss = confidenceLoss + \alpha * locationLoss \tag{6}$$

Where the α value is used for weighing and is tuned such that we can minimize our overall loss. For three, the algorithm does calculations based on what features it identified and will attempt to make a prediction of what the object is and what the probability is that it is that object. For four, the algorithm will attempt to fix its weighing of values such that the loss is minimized for future iterations.

Verification

The model overall was fairly accurate overall. Class accuracies for the entire model are reported in Appendix A. While the goal of 70% accuracy was not obtained in practice, the accuracy reported by the model creators was found to be accurate over the course of experimentation. We placed the goal object at sets of positions with respect to the camera and tracked whether detection occured.

Object Retrieval

Distance Sensors

Design

We opted to use ultrasonic sensors because of their low cost as compared to other rangefinding options like lidars or other laser based sensors. We mounted them across the front of the robot to get a spread of distance readings from the surrounding environment. The sensors send out ultrasonic pulses and return the length of time for the sensor to hear the echo. Using Equation 7, we were able to calculate the distance to the object.

$$Distance(m) = Time(\mu s) * \frac{1}{2} * 0.000343(\frac{m}{\mu s}) \tag{7}$$

We found these values to be quite noisy and decided to take an average of consecutive readings in an attempt to reduce the spread. Unfortunately we found that our ultrasonic sensor output was still very noisy even within the primary operating range. We needed a large perpindicular surface for the sensor to return an accurate reading and the objects that were within the grasping capabilities of the manipulator did not meet that requirement for the most part.

Verification

The accuracies of the ultrasonic sensors at operating ranges are given in Appendix A. Beyond these ranges, the perpendicular profile of the bottles were too narrow to be properly detected, and the detection algorithm also struggled to identify the bottle.

Manipulator

Design

We chose a very simple manipulator for this application, attaching a prefabricated gripper to servo oriented such that grasped objects can be lifted off of the ground. Control of the servos are run directly from the Raspberry Pi using a PWM output from the GPIO pins. We chose the servos based on the available torque, which needed to be able to lift the gripper, the gripper servo, and the object itself. The gripper fit well within the robot dimensions and allowed for a five centimeter wide object to be picked up.

Verification

The lifting capabilities of the manipulator were checked by grasping and lifting the goal object for a sustained period, to simulate carrying it back to the end goal position.

Retrieval

Design

The object retrieval protocol began when the object detection algorithm located on object of the goal class. It can be broken down into three steps. First the robot rotates towards the object until it is within a angular tolerance value. It calculates the angle to be turned through by calculating the distance from the pixel centerline to the centroid of the bounding box drawn for the object. It then scales that against the field of view of the camera. Figure 7 shows an illustration of the camera frame.

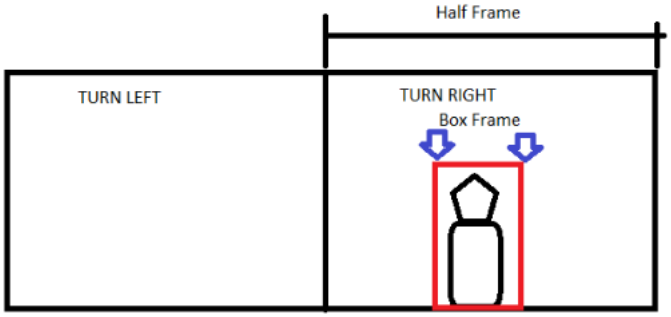

Figure 7: Retrieval Rotation Visualization

Once the robot has successfully turned to the object, it uses the distance measurement returned by the forward facing ultrasonic sensor to move forward to the appropriate distance from the object to deploy the gripper. When the robot is in the gripping position, it lowers the manipulator and graps the object, then lifts it so that it is not dragging on the ground.

Verification

Succesful completion of this task was hampered by three factors. The lower classification accuracy for bottles meant that a large fraction of the time, the robot would not detect the object and the retrieval protocol would not be initiated. Secondly the large moment of inertia of the robot meant that it was not able to complete small turns that were required when the bottle was placed close to the camera centerline. Lastly, the inaccurate values reported by the ultrasonic sensors often caused the robot to drive through the goal object because it expected to be further away. We were not able to achieve consistent success with the object retrieval, but we could show that the components involved performed reasonably well individually.

Cost

Parts

Table 1: Parts Costs

Part	Manufacturer	Retail Cost ($)	Quantity	Shipping ($)	Actual Cost ($)
Raspberry Pi	CanaKit	42.99	1	0	42.99
99:1 Gearmotor	Pololu	34.95	2	2.37	72.27
47:1 Gearmotor	Pololu	34.95	2	0	69.90
Mounting Hub 3M	Pololu	6.95	1	0	6.95
Mounting Hub 4-40	Pololu	6.95	1	0	6.95
HC-SR04 Ultrasonic	Keywish	10.59	1	0	10.59
32 GB MicroSD Card	Sandisk	11.99	1	0	11.99
Raspberry PiCam V2	Amazon	25.55	1	0	25.55
RaspiCam 24” Cable	Adafruit	7.23	1	0	7.23
HS-422 Servo Motor	RobotShop	11.49	2	9.00	31.98
Large Robot Gripper	RobotShop	19.50	1	0	19.50
DRV8835 Motor Driver	TI	1.43	16	29.62	58.17
DRV8835 Driver Chip	Pololu	3.89	5	3.95	23.40
Gearmotor Bracket	Pololu	7.45	1	8.95	16.40
2200 mAh 3S Lipo	Turnigy	8.79	2	8.10	25.68
Wheel Pair 70x8mm	Pololu	8.49	1	8.95	17.44
Multipurpose Bracket	Pololu	11.95	1	9.00	20.95
”C” Servo Bracket	Pololu	7.95	1	0	7.95
PCB	PCBWay	20.00	5	0	100.00
Total					575.89

Labor

This project was the culmination of many hours of work amongst the group. We also asked the ECE Machine Shop to fabricate and assemble the frame of the robot. We estimate that the cost of the required labor for this project can be estimated with the following equations.

$$675 \text{ hours} * 35 \frac{\$}{\text{hour}} * 2.5 = \$59062.5 \tag{8}$$

$$11 \text{ hours} * 35 \frac{\$}{\text{hour}} * 2.5 = \$962.5 \tag{9}$$

Equation 8 gives the cost estimate of the groups' collective labor. Equation 9 follows the same formula for calculating labor value. The hour value is an estimate of how long it took our contact at the machine shop to fabricate all of the components. Combined, these sum to a total labor cost of $60025.

Conclusion

Accomplishments

This project produced a fully ROS integrated differential drive robot that is nearly capable of using an advanced machine learning algorithm to identify and retrieve target objects. We were able to show proper functionality of the single shot detection algorithm for finding and correctly localizing several classes of objects. The full retrieval operation was shown to work in parts, from locking on to picking up the object. While struggling to cooperate with the object detection, the navigation system was also shown to work to a degree. Its performance was limited by several factors, but it still consituted a fully functional differential drive robotic platform.

Ethical considerations

We believe that our project is aligned with the first tenet of the IEEE Code of Ethics, to hold paramount the safety, health, and welfare of the public, [4] because our project is designed to help move robotic understanding of real world systems towards the ability to save lives. We strive to use the understanding of intelligent systems to benefit the public good. This leads to the importance of #5 of the Code, to improve the understanding by individuals and society of the capabilities and societal implications of conventional and emerging technologies, including intelligent systems [4] in that it will be our duty to inform the public about the beneficial uses of the technology that we are working with and how they can be further used to help society. Since the success of the project is directly dependent on the functionality of the reinforcement learning algorithm, it is very important that we accurately report our results, regardless of the outcome. Inconsistent data and unreliable reporting would violate #3 of the Code [4] and would negatively impact the field of robotics research and our character as engineers. In the same vein, it is very important that we give appropriate credit for the previous works that we use and build on to develop our system. It would be unethical to take credit for the work of others in accordance with #7 of the Code [4]. We will be using and learning from many different research sources as well as from our peers and faculty members as we progress through this project and need to accurately present the chain of knowledge and development.

Future work

The major obstacles that affected the robot's performance were the large weight of the frame, the inaccuracy and limited angular resolution of the rangefinding system, and the Raspberry Pi's limited and non-realtime processing capabilities. These will be the first obstacles to overcome.

The frame of the robot requires a far lighter material and possibly a redesign of the layout to minimize the weight, while still maintaining functionality. The camera, rangefinder, and the manipulator all need to be forward mounted and must share vertical real estate, so any potential redesign must account for their collective placement.

The ultrasonic sensors produced a large amount of noise and were not able to accurately range objects of the size and perpendicular profile that we were working with and could be picked up by the manipulator. This damaged the performance of both general navigation as well as object retrieval. The expectation of

ROS robots using the full navigation stack is that they are equipped with a lidar sensor, and it would be prudent to upgrade to a reasonably priced laser-based rangefinding system.

The Raspberry Pi struggled to handle the realtime nature of the signal handling for the hardware components of the robot in addition to doing computationally heavy machine-learning computer vision. It is the recommendation of experts that for controlling robots with a Raspberry Pi, an additional microcontroller is used to handle the peripherals while the Raspberry Pi is used for high level control. A Pi Hat with an ATMega processor has been purchased for this purpose with which the Pi can share the load. It will also be useful to add an inertial measurement unit in order to better localize the robot.

After these hurdles are dealt with, the goal will be to accomplish the original goal of the project, implementing a reinforcement-based exploration of simple environments. This project has the potential to be useful in many future applications.

References

[1] D. Nosowitz, "Meet japan's earthquake search-and-rescue robots," 2011. [Online]. Available: https://www.popsci.com/technology/article/2011-03/six-robots-could-shape-future-earthquake-search-and-rescue

[2] L. D'Monte, "5 disaster robots that may rescue you from natural disasters," 2015. [Online]. Available: http://www.govtech.com/em/safety/5-Robots-That-May-Rescue-You-From-Natural-Disasters.html

[3] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *CoRR*, vol. abs/1602.01783, 2016. [Online]. Available: http://arxiv.org/abs/1602.01783

[4] IEEE, "Ieee code of ethics," 2017. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html

Data Tables

Distance Sensor Data

Table 2: Ultrasonic Sensor Accuracy

Actual Distance (cm)	Meas 1	Meas 2	Meas 3	Meas 4	Meas 5	Average	Error
5	6.42	6.41	6.41	6.41	6.42	6.414	28
10	12.058	12.07	12.12	12.05	13.64	12.387	23.876
15	15.905	15.967	15.943	16.02	15.988	15.965	6.43
20	20.78	20.78	20.67	20.66	20.67	20.712	3.56
25	25.667	25.605	25.649	25.637	25.625	25.583	2.33

Object Detection Data

Table 3: SSD Class Detection Accuracies

Object	Accuracy	Object	Accuracy
Aeroplane	0.8739	Dining Table	0.435
Bicycle	0.4864	Dog	0.673
Bird	0.7727	Horse	0.5575
Boat	0.75	Motor Bike	0.5023
Botlle	0.5706	Person	0.821
Bus	0.614	Potted Plant	0.7723
Car	0.6823	Sheep	0.8661
Cat	0.7598	Sofa	0.9067
Chair	0.332	Train	0.9133
Cow	0.6408	TV Monitor	0.7107
		Overall	0.7177

Differential Drive Data

Table 4: Motor Angular Speed Response

Angular Velocity (rad/s)	Lower Bound (rad/s)	Upper Bound (rad/s)	Range (rad/s)
1	1.73	0.32	1.41
2	2.69	1.34	1.35
3	3.1	2.91	0.19
4	4.13	3.85	0.28
5	5.15	4.85	0.3
6	6.21	5.79	0.42
7	7.28	6.79	0.49
8	8.29	7.7	0.59
9	9.3	8.63	0.67
10	10.38	9.72	0.66
11	11.35	10.5	0.85

Table 5: Differential Drive Wheel Response

Input		$\omega_r(\frac{rad}{s})$		$\omega_l(\frac{rad}{s})$		**Input**		$\omega_r(\frac{rad}{s})$		$\omega_l(\frac{rad}{s})$	
$v(\frac{m}{s})$	$\omega(\frac{rad}{s})$	Expected	Actual	Expected	Actual	$v(\frac{m}{s})$	$\omega(\frac{rad}{s})$	Expected	Actual	Expected	Actual
0.1	0	2.857	2.78	2.857	2.78	0.1	1.0	5.536	5.55	0.179	X
0.2	0	5.714	5.62	5.714	5.62	0.1	3.0	10.893	10.87	-5.179	-5.20
0.3	0	8.571	8.58	8.571	8.58	0.1	-1.0	0.179	X	5.536	5.56
0.4	0	11.428	11.41	11.428	11.41	0.1	-3.0	-5.179	-5.21	10.893	10.91
-0.1	0	-2.857	-2.8	-2.857	-2.8	0.2	2.0	11.071	11.24	0.357	X
-0.2	0	-5.714	-5.56	-5.714	-5.56	0.2	-2.0	0.357	X	11.071	11.27
-0.3	0	-8.571	-8.48	-8.571	-8.48	0.3	1.0	11.25	11.38	5.893	6.05
-0.4	0	-11.428	-11.31	-11.428	-11.31	0.3	-1.0	5.893	6.14	11.25	11.43
0	1.0	2.687	X	-2.687	X	-0.1	1.0	-0.179	X	-5.536	-5.45
0	2.0	5.357	5.44	-5.357	-5.32	-0.1	3.0	5.179	5.53	-10.893	-10.74
0	3.0	8.036	8.11	-8.036	-7.92	-0.1	-1.0	-5.536	-5.25	-0.179	X
0	4.0	10.714	10.77	-10.714	-10.58	-0.1	-3.0	-10.893	-10.61	5.179	5.48
0	-1.0	-2.687	X	2.687	X	-0.2	2.0	-0.357	X	-11.071	-11.01
0	-2.0	-5.357	-5.38	5.357	5.46	-0.2	-2.0	-11.071	-10.94	-0.357	X
0	-3.0	-8.036	-7.93	8.036	8.11	-0.3	1.0	-5.893	-5.64	-11.25	-11.01
0	-4.0	-10.714	-10.7	10.714	10.77	-0.3	-1.0	-11.25	-11.07	-5.893	-5.71

Requirement and Verification Table

Table 6: System Requirements and Verifications

Requirement	Verification	Verification status (Y or N)
1. Object Detection Requirements (a) Object detection is accurate to within 70 percent. (b) Object centroid placement within .5 cm.	1. Verification (a) Test algorithm on VOC 2010 image set (approx 10,000 images). Check the prediction accuracy by keeping track if the model properly classifies the object. (b) Get the frame size from the object detection program that the robot is running. This corresponds to the size of the camera image. Divide it into the right and left image halves each corresponding to a turning direction. The range of degrees for the camera is at max 31.1 degrees, so each frame unit corresponds to 0.155 degrees. Using the inverse tangent we need a distance of at least 2 meters for 0.5 cm centroid error. This assumes a perfect placement model.	
2. Maze Solving Requirements (a) The A-star algorithm outputs the optimal path through the created maze, from the start position to the goal position. (b) The robot uses reinforcement learning model to navigate from the start to the object location.	2. Verification (a) Run the maze under a Breadth-First Search (BFS). BFS is guaranteed to give an optimal path on the one start one end maze that we are using. So, comparing the output of the A* to the BFS can give an idea about how optimal it is. (b) Monitor the decision making of the reinforcement learning algorithm at each state of the process to show that it is using a model to make decisions. Monitor that the robot is able to explore a maze environment without hitting the environment.	

Continued on next page

Table 6 – continued from previous page

Requirement	Verification	Verification status (Y or N)
3. Navigation Requirements (a) The robot is able to move within 5 cm for a 30cm straight linear motion. (b) The robot is able to move within 5 degrees for a 90 degree turn in either direction.	3. Verification (a) Use a tape measure to show the distance to be traveled by the robot. Publish a movement goal of 0.3 in the x direction with the identity quaternion. Calculate the physical error and compare to the odometric position of the robot as reported by the system. (b) Use a 90 degree angle on the floor, either drawn on paper or a tile corner. Place the robot on the vertex of the angle aligned with one of the rays. Publish a movement goal of the 90 degree rotation equivalent quaternion. Calculate actual turned angle and compare to the reported angle of the system.	

Continued on next page

Table 6 – continued from previous page

Requirement	Verification	Verification status (Y or N)
4. Driving Requirements (a) The motors can be driven at specific angular velocities up to 13 radians per second with a 0.2 radians per second noise tolerance. (b) The motor step response has less than 10 percent overshoot. (c) The motor step response has less than 0.1 second rise time. (d) The motors can be driven in tandem using a set linear and angular velocity input. (e) The encoder reading, control calculation, and motor output control can all be run at least 50 Hz.	4. Verification (a) Output desired angular velocity for the wheels using the ROS message publishing functionality. Plot the angular velocity of the wheels as reported by the motor encoders. Check to see that the current angular velocity matches the desired. Test at integer values of angular velocity up to 13 radians per second. (b) Use the ROS plotting functionality to visualize the motor step response. Plot the angular velocity of a wheel, and input the desired wheel velocity. Wait for the wheel velocity to stabilize, then increase the desired wheel velocity by 1 radian per second. Pause the plot to see the response. Calculate the relevant parameter of the response. (c) Use the ROS plotting functionality to visualize the motor step response. Plot the angular velocity of a wheel, and input the desired wheel velocity. Wait for the wheel velocity to stabilize, then increase the desired wheel velocity by 1 radian per second. Pause the plot to see the response. Calculate the relevant parameter of the response. (d) Use the differential drive equations for wheel velocities to calculate the correct angular velocity of each wheel. Compare to the average value of each motor as reported by the system. (e) Use the ros publishing rates to verify the speed of each type of function.	

Continued on next page

Table 6 – continued from previous page

Requirement	Verification	Verification status (Y or N)
5. Sensor Requirements (a) The ultrasonic sensor returns valid distance readings to within 10 percent of the measured actual value for valid measurement ranges, between 20 and 50 centimeters. (b) The pseudo-laser sensor correctly integrates and outputs the values of the separate ultrasonic sensors. (c) Subrequirement	5. Verification (a) Set the robot along a tape measurement such that the ultrasonic sensor speakers line up with the origin of the line. Place a flat object at 5 cm intervals and calculate the error of the measurement output by the system. (b) Calculate the origin of the pseudo-laser sensor by finding the intersection of the normals to the ultrasonic sensor boards above the body of the robot. Place objects in front of all three ultrasonic sensors in the valid range of 20 to 50 centimeters and measure the distance from the laser origin point to the objects. Calculate the measurement error. (c) Subverification	
6. Hardware Requirements (a) 6V Voltage regulator must supply a constant 6V $\pm$ 5 % output voltage while the battery discharges. It must be able to sustain currents up to 1.8 amps. (b) 5V Voltage regulator must supply constant 5V $\pm$ 5 % output voltage while the battery discharges. It must be able to sustain currents up to 1 amp.	6. Verification (a) Measure the voltage output of the regulator over the course of a gully charged battery operating discharge. Clamp the motors and drive until the motor drivers engage over current protection shutdown. Measure the amperage with a multimeter. (b) Measure the voltage output of the regulator over the course of a gully charged battery operating discharge. Measure the amperage with a multimeter when all peripherals are connected to the Raspberry Pi and running.	

Continued on next page

Table 6 – continued from previous page

Requirement	Verification	Verification status (Y or N)
7. Gripper Requirement (a) The grasping servo must provide at least 1 N*m to grasp object to overcome the gravitational pull. (b) The lifting servo must provide at least 1.5 N*m to lift the gripper, grasping servo, and the object.	7. Verification (a) Grip object above ground level to check that it doesn't slip. (b) Pick up object with gripper assembly to test the ability to lift goal objects.	