SOS_bot

Author: Karen Johnson, Manini Amin, and Joseph Wang
Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*— A system capable of** assessing the initial person count in a disaster zone. The SOS_bot enters areas affected by disasters such as earthquakes or fires and provides first responders with information to aid in the supplies and number of personnel needed to treat the situation. A combination of autonomous path planning, obstacle avoidance and machine learning algorithms, the SOS_bot is able to detect and report the amount of people in a room so that human risk is limited in the initial stages of disaster recovery.

***Index Terms*—Create2, Faster RCNN, Ultrasonic sensors, Path Planning, Obstacle Avoidance, Human detection, Grid System, COCO dataset, Pytorch**

I. Introduction

During the initial stages of a disaster situation, safety is the primary concern, for both the victims of the situation and the first responders. In order to aid in the efforts to minimize risk for both sides, the SOS_bot has been created to provide immediate disaster recovery information. Currently, when a scenario as described before occurs, first responders must enter the zone without any information on what the present situation looks like other than aerial shots from an overhead helicopter. Often times, without any information these first responders find themselves with either a severe lack of medical equipment needed to treat victims or underestimation of the personnel and/or tools needed to deal with the situation. With SOS_bot however, first responders will be able send the bot in to the zone beforehand so that it will autonomously traverse the area and provide a total count on the victims present for responders to then adjust with.

The SOS_bot must be able to autonomously navigate the space, arriving to the three entered points of interest within a margin of 0.5 feet. The bot must ensure complete avoidance of any obstacles in its path, with a berth of 0.5 foot between itself and the object- accounting for the case that the obstacle is an injured human. Still photos must be taken with 360 degree coverage of the room. Once the pictures have been sent to the responder's local computer, the SOS_bot UI must report the count with a detection accuracy of at least 55% after using the Faster RCNN model. The bot must give a clear picture of the situation at hand to the responders.

II. Design Requirements

The SOS_bot will be evaluated on an obstacle course that is design and built by us. The obstacle course will be a 10 foot by 10 foot marked space filled with randomly sized obstacles as well as people. The obstacles will be of size 1 foot by 1 foot. Their heights will be no taller than 2 feet. People will be scattered around the course for the robot to detect.

The robot will be given the layout of the area, including where it will be starting, the boundary of the space, as well as the key locations it must go to to take panoramic views. The robot will not have any information of any obstacles that may potentially be between where it is and the destination. It must be able to initially calculate the most efficient path between all of the points of interests and arrive at each point with an error margin of 0.5 feet. If the robot encounters an obstacle along the way, it must not make contact with said obstacle but instead move around it with at least 0.5 feet of clearance when possible. Once it reaches it destination, it must rotate in a way such that the camera is able to take pictures to cover a 360 degree view from where it is. These images will be sent back to a local computer via wifi for further evaluation.

On the local computer, there will be a user interface that sends the images to the cloud where a deep learning model that is trained on the COCO dataset for people will be. The model must achieve a mAP score of 55% on the COCO dataset for people detection before it is deployed on our machine. Furthermore, it should be biased towards false positives. When running inference on the images taken by the robot, no more than 10% of the detections should be false positives and no more than 5% of the detections should be false negatives. The model should be able to detect people in rooms with poor and varied lighting. Lastly, the model should be able to detect people even if only partial views of the person are given. These partial views must have a major identifying body part such as an arm, hand leg, or face. Once the computer has finished detecting the images it must be able to display it's findings on a user interface at 1080p resolution.

III. Architecture and/or Principle of Operation

The SOS_bot will consist of 2 major components. The first component is the physical robot itself. It will contain a camera, sensors, and an iRobot that are all connected to a Raspberry Pi. The robot will be handling any movement or tasks in the obstacle course. This includes path planning, obstacle avoidance, and picture taking. It will also handle sending images over to the local computer.

The second major component is the people detection deep learning algorithm running. The Faster RCNN model will be trained on a GPU machine on AWS. It will be trained using the COCO dataset, which contains several hundred thousand images. When the images from the robot are transferred to the local computer, they will be sent to the cloud. This is where inference will take place and where the bounding boxes on the images will be drawn. These detection images will then be transferred back to the local computer to be displayed on the UI for a given point of interest. This user interface allows people to select where they want the bot to go to as well as see the results of the inference.

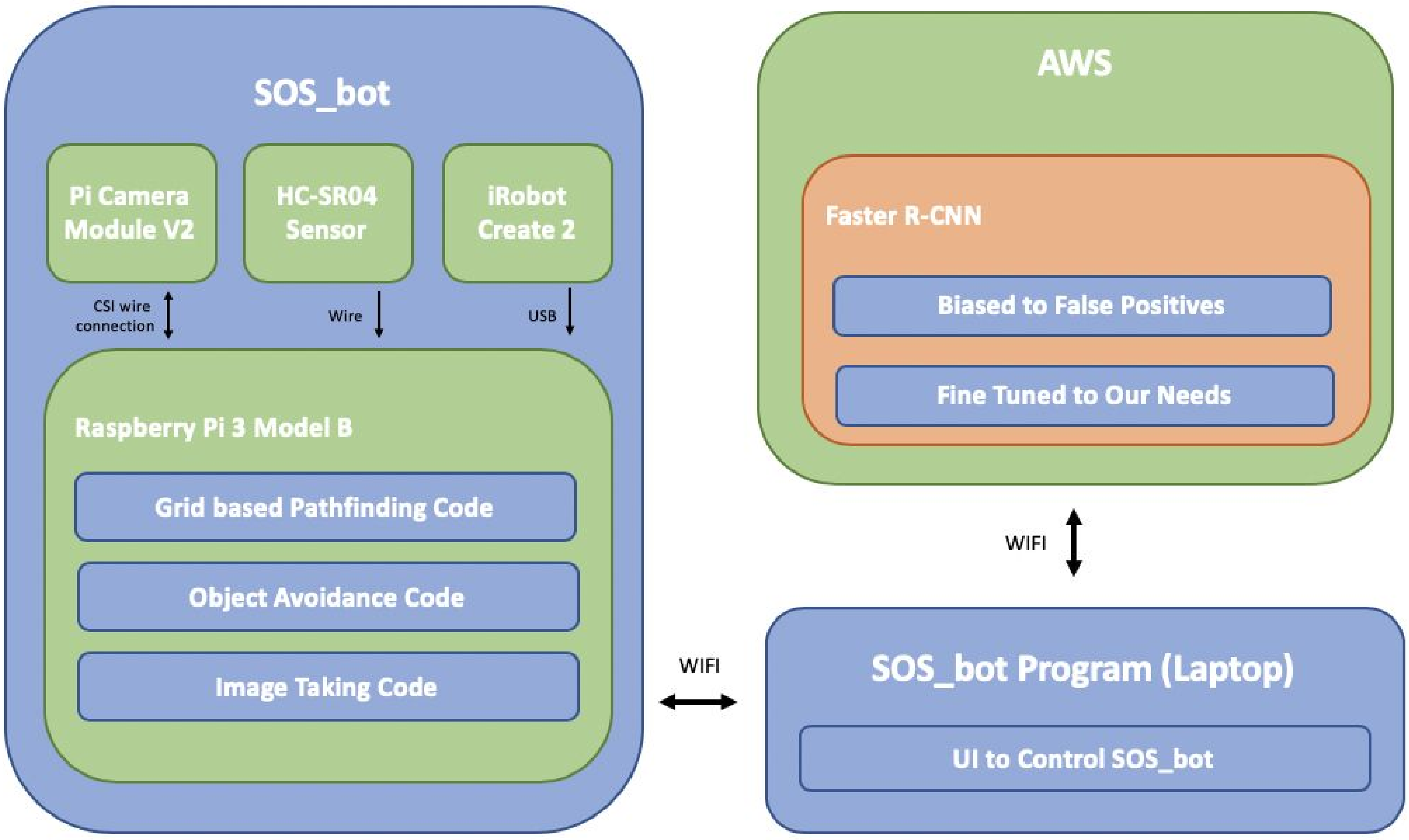
Figure 1: System Architecture

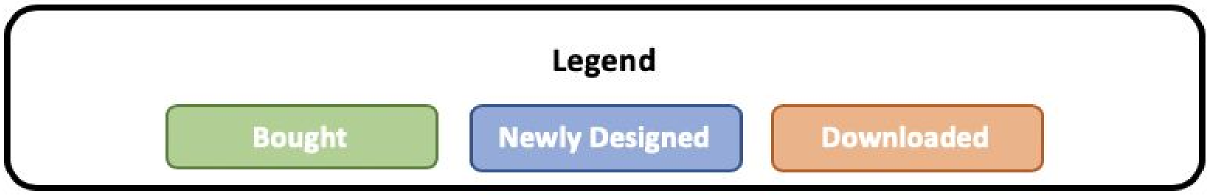
IV. Design Trade Studies

The SOS bot incorporates hardware and software decisions made after assessing the tradeoffs of multiple approaches for each component. These considerations were necessary to ensure that the SOS bot detects the number of humans in a disaster zone in an efficient, accurate, and practical way. These tradeoffs will be discussed in the sections below.

A. Robot Design Tradeoffs

A mini drone and an iRobot Create 2 were the two major components that we were looking at for the base of our robot. Both have clear benefits and drawbacks. The drone is the more practical choice in terms of its mobility and fit for the use case of our robot. Rescue teams would mostly likely not send in a slow moving vacuum cleaner into a disaster area to look for humans. Drones would also offer us movement along 3 axis, making navigation through an obstacle course much easier. It would also allow us to have a better angle for pictures as we can just fly up to a proper height before taking an image





instead of being stuck on the ground. The main issues with drones was that most drones on the market were not designed to carry much weight nor are they able to fly for very long. Furthermore, these drones usually do not come with libraries that allow us to control. The drones that did have the capabilities we were looking for cost well over our budget. When reaching out to researchers in the Robotics Department, we found that most of them built their own drones. This would make the scope of our project too large.

Ultimately, we decided that even though the iRobot Create 2 would not be practical in a real search and rescue mission, it would still be a good base for our demo as it had the battery life as well as the movement libraries that we desired. We would also not have to assemble the Create 2 ourselves like we would have most likely for the drone and we also have the added bonus of not being concerned that the Create 2 crashes and damages itself since it is quite a slow moving machine. This could potentially save us some more money from our budget as we would not have to purchase any replacement parts.

A. *Human Detection Model Design Tradeoffs*

YOLO and Faster RCNN are two of the most popular deep learning models used for object detection. Faster RCNN is a region based detection algorithm that utilizes a region proposal method to generate regions of interest that are later passed through fully connected layers for localization and classification. YOLO, You Only Look Once, is a single shot detection algorithm that predicts both the boundary box and the classification at the same time with one Convolutional Neural Network. This is in comparison to Faster RCNN which requires a region proposal network followed by a convolutional neural network.

The YOLO model has a low inference time and has the ability to achieve real time object detection. However, one of its limitations is the accuracy it can produce. As show in (1), YOLOv3-608 achieves a mAP (mean Average Precision) score of 33% when trained on the Microsoft COCO dataset [1]. However, the model has an inference time of 51 ms and can process a maximum of 91 frames per second [1]. In comparison, Faster RCNN has a limitation of processing 17 frames per second, but achieves a mAP score of 34.9 % [1].

Single shot detection models, specifically YOLO, cannot beat the accuracy that the Faster RCNN model can produce. Although deeper analysis indicates that the accuracy advantage Faster RCNN gives is usually not worth the significant sacrifice in speed that occurs, our SOS_bot will utilize the region-based Faster RCNN model. This decision was made for the simple reason that we are not aiming to achieve real time detection. Thus, the slight percentage increase in accuracy is more valuable to our system. The SOS_bot's main goal is to provide accurate and reliable information to first responders. As real time detection is not a goal the robot must achieve, the sacrifice in speed is worth the 2% increase in accuracy. This design decision ensures that our SOS_bot is designed with the application area in mind. This model best aligns with the project's goals of providing a reliable count of the number of humans trapped in a disaster zone.

After determining which model would best align with our target use case, model training trade-offs were considered.
First, the dataset we wanted to use for training was determined. Two popular datasets used for object detection algorithms are the PASCAL VOC and COCO datasets. The COCO dataset (19GB train/val) is significantly larger than the PASCAL VOC dataset (450MB train/val). Since the COCO dataset is larger, the variation in images is higher which leads to better and more robust model training. However, this larger dataset comes at a price. Not only does this lead to a huge increase in training time it also requires more money to support the increased memory and training time required on AWS resources. When training this faster RCNN model on a P2.8xlarge (8 GPU optimized EC2 AWS machine) with the COCO dataset, the average training time per epoch is 6.0 hours. On the other hand with PASCAL VOC, this time drops to .17 hours. In the end, we made the decision to allocate more space in our budget for more robust and accurate training.

One final design tradeoff we made was whether to run the model on a local computer or on the cloud. At first we wanted the the model to be trained on the cloud and run on the computer but after further evaluation we found it very difficult to get the model on to a local computer and it's speed would be extremely slow. We ultimately decided that it would make more sense to run the person detection model on the cloud.

Table 1

Model	**mAP Score (mean Average Precision)**	**FPS (Frames per second)**
Faster RCNN	34.9%	17
YOLO	33%	91

Figure 2: Models Trained on MS COCO dataset

V. System Description

The SOS_bot is comprised of two major subsystems. The first subsystem constitutes the robot itself that moves around the designated room and takes the still pictures of the space. The second subsystem is related to the machine learning model that examines the pictures taken and details the human count to the interactive user interface.

A. Subsystem A

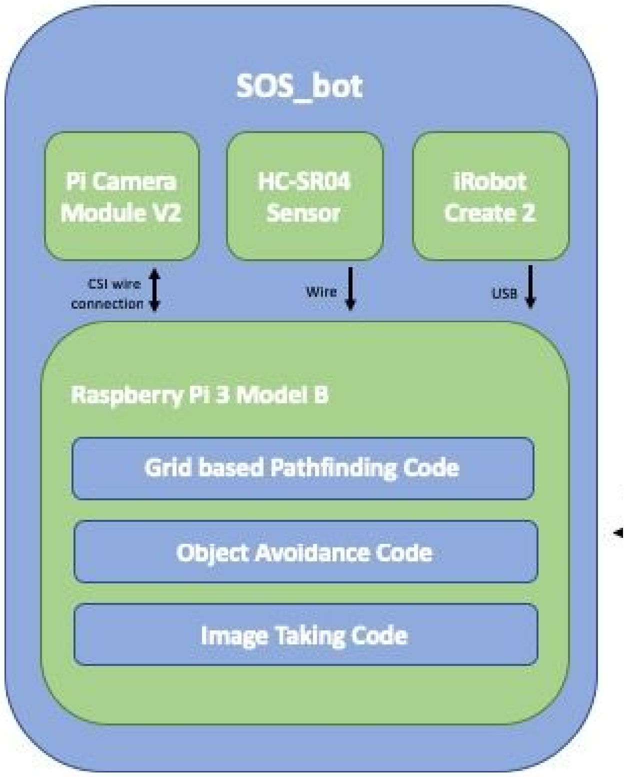


Figure 3: Subsystem A

As shown in the diagram above, subsystem A is comprised of the iRobot Create2, the ultrasonic sensors, a pi compatible camera and the Raspberry Pi 3 itself. The pi will be serially sending commands to the Create2 using a USB cable and connected to the camera via the camera port. The ultrasonic sensors are connected via the GPIO pins.

The SOS_bot takes in the coordinates from that the user interface sends via SSH and precalculates the most optimal path between points. Then, the robot moves at a set velocity for a the correct amount of time to reach each point. Angles are also calculated in a similar fashion. We set a predetermined velocity for each wheel of the iRobot, one positive and one negative, and this allows us to turn in place. Once the robot completes movement at an angle, we always readjust to forward. This is to negate an accumulation of error due to inaccurate movement. During the movement phase, the robot is constantly checking the ultrasonic sensors for any potential obstacles that it may encounter.

For obstacle avoidance, we have two ultrasonic sensors pointing forward, one on the left side of the robot and the other on the right. This allows us to choose the most efficient way to avoid anything we encounter. If the left sensor picks up an obstacle the right sensor does not, we turn right. If the right sensor picks up an obstacle and the left one does not, the robot turns left. Finally, if both sensors pick up an obstacle, we default to turning right.

When the robot reaches an interest point, the robot reorients itself to facing forward and begins to take 6 pictures. Between each image the robot turns 60 degrees. This is to so that we can create the panoramic view that we want. Once the images are taken, the robot does a 360 degree turn in the opposite direction to eliminate any accumulated errors and continues to it's next point.

B. Subsystem B

The second subsystem of the SOS_bot is the program that runs on the laptop as well as on Amazon Web Services. The program consists of two parts: the user interface to control the SOS_bot along with the deep learning algorithm the does the detection on the images. The user interface is written using python and the tkinter library. It consists of a grid layout of the obstacle course that users can place up to 3 interest points for the robot to go to. Once users have selected the points of interests that they want to go to, they can press start and the program will upload the map along with the interest points onto the robot via SSH through WIFI. The user interface then waits for the robot to arrive at the interest points and take images. Once the images are taken, the user interface copies them back on to the laptop via SSHFS. It is worth noting that this is done as soon as the robot reaches an interest point so users do not have to wait for all interest points before receiving the images.

Once the program receives the images, it send them to the an AWS cloud instance through WIFI. The instance runs the images through the second part of the subsystem: the Faster RCNN deep learning model. The model is implemented using Pytorch and trained on a subset of the COCO dataset, specifically the one that contains images of people. The model outputs a binary classification of the presence of humans in the image along with the estimated bounding box of where that human is. We retrained the Faster RCNN model to perform binary classification in place of multiway classification. The model is biased towards false positives for human detection and runs on a p2.8xlarge system on AWS and will show where any humans are in the images that the robot sent. The biasing was incorporated into the Cross Entropy Loss function for the classification neural network. The bias was a parameter that penalized false negatives more heavily than false positives. We trained the model for 3 epochs with a batch size of 16 and a learning rate of .001. The network architecture we used was res101. Once the model inference is done, the new images with bounding boxes are sent back to the laptop. The user interface then stitches the images together using the Python Image Library and display the panoramic for users to see.

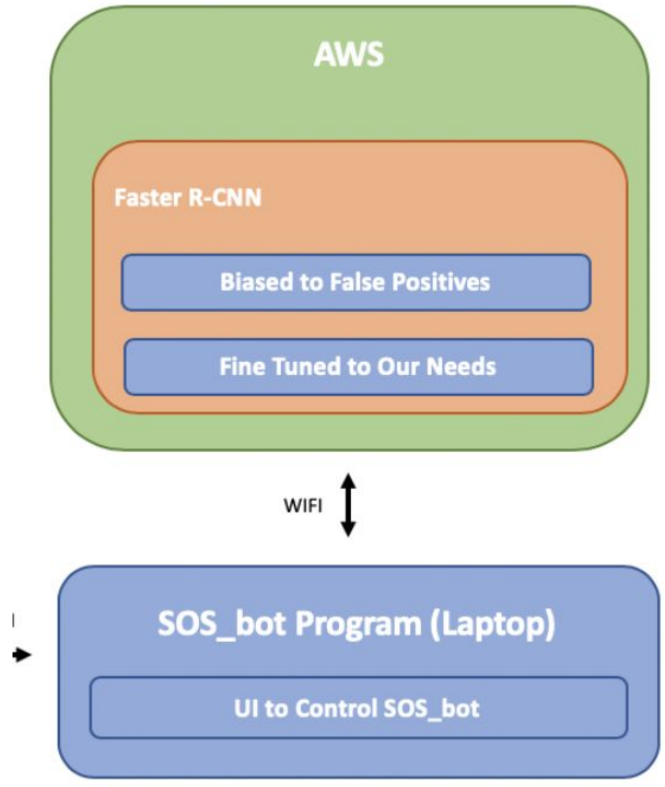


Figure 4: Subsystem B

V. Project Management

A. Schedule

The schedule that we ended up with was somewhat different from the schedule that we started with. As SOS_bot was worked on, the areas that are more time consuming have become more apparent and the schedule has been adjusted accordingly. An area that was initially underestimated was the time spent waiting for parts to arrive. This delay took longer than the week of slack that was allocated for delivery. Thus, the firmware for the bot has been pushed farther along as the hardware components were necessary for testing was not present.

We also spent quite a bit of time on refining the movement of the robot. This is because while there exists a library for movement with the iRobot, we found that it was not up to our specification in terms of our accuracy. Therefore, Karen ended up having to write her own library and finetune the velocity and time traveled to get the accuracy that was desired. This caused significant delays in or schedule, forcing us to move some of our tasks around.

One task that we moved due to the delays caused by movement was the user interface. We also originally planned for the user interface to be developed towards the end of the project but because Joseph could not work with the Raspberry Pi on obstacle avoidance while Karen was writing a new library for movement, he started to work on the user interface early. This allowed us to stay relatively on schedule to finishing by the end of the semester but also allowed us to test integration early.

Once movement was finished, we moved on to obstacle avoidance. We spent a few weeks on implementing and refining this and by mid-April we were able to have a fairly accurate movement and obstacle avoidance implemented. Once movement was at a place that we wanted, we spent a week integrating everything together.

The development of the detection model happened in parallel to the hardware portion of the SOS_bot. We spent a lot of time in the beginning researching the best approaches to human detection and decided on the Faster RCNN model. From there, we spent a lot of time on implementing, biasing, and training the model. By the first week of April, however, Manini was able to have a preliminary model setup and outputting what we wanted. From there, she kept fine tuning the model and implementing the pipeline of sending and receiving the images to and from the cloud.

After all the subsystems were implemented, we began to integrate everything with the user interface. Our schedule allowed us to stagger the ending of movement and model development approximately 2 weeks apart. This allowed ample time to test the integration of movement with the user interface before having to integrate the model. Once everything was integrated, we did one final week of testing before out public demonstration.

We were able to mostly stick to our schedule throughout the semester with ample amounts of testing individual subsystems as well as the full system. While some things took longer than we thought they would, the slack weeks that we planned in our original schedule allowed us to make up for the time lost and finish the project on time.

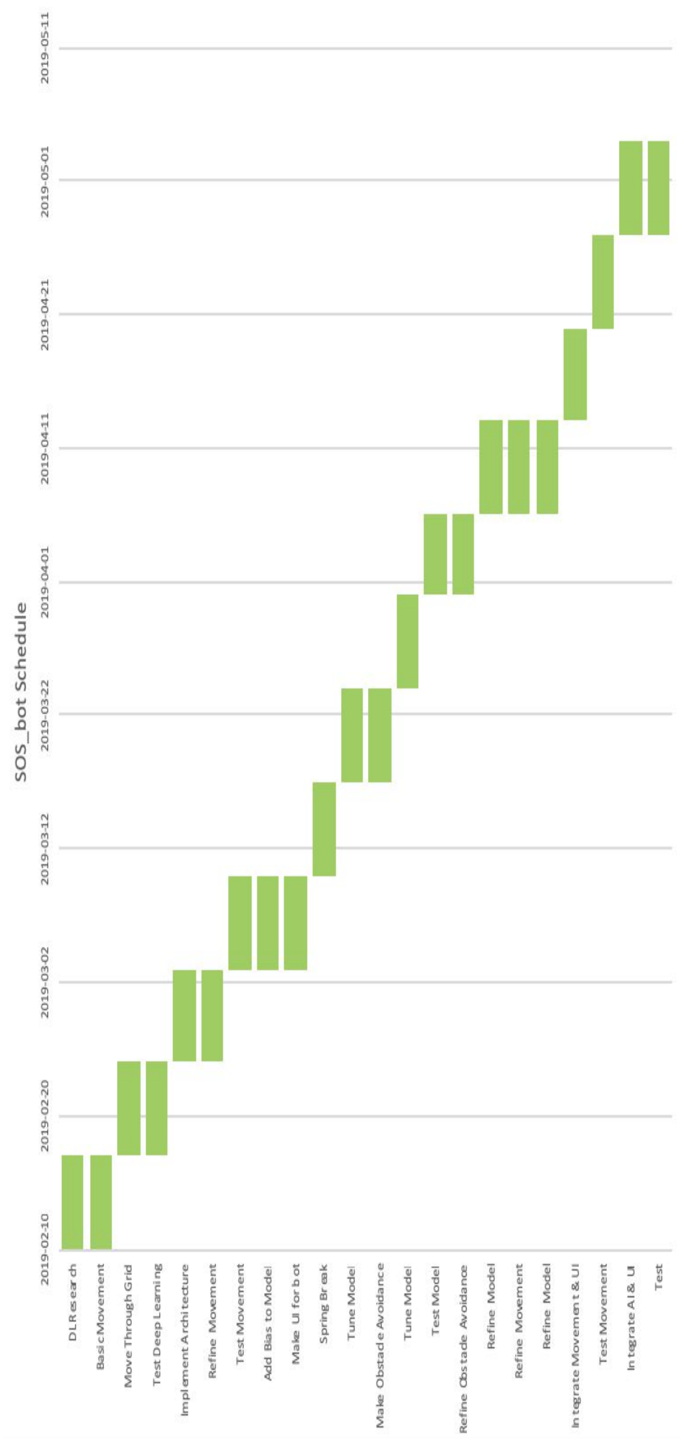


Figure 5: Gantt Chart

TASK NAME	ASSIGNED TO	START DATE	DUE DATE
DL Research	Manini & Joseph	2/10/19	2/17/19
Basic Movement	Karen	2/10/19	2/17/19
Move Through Grid	Karen	2/17/19	2/24/19
Test Deep Learning	Manini & Joseph	2/17/19	2/24/19
Implement Architecture	Manini & Joseph	2/24/19	3/3/19
Refine Movement	Karen	2/24/19	3/3/19
Test Movement	Karen	3/3/19	3/10/19
Add Bias to Model	Manini	3/3/19	3/10/19
Make UI for bot	Joseph	3/3/19	3/10/19
Spring Break	All	3/10/19	3/17/19
Tune Model	Manini	3/17/19	3/24/19
Make Obstacle Avoidance	Joseph & Karen	3/17/19	3/24/19
Tune Model	Manini	3/24/19	3/31/19
Test Model	Manini	3/31/19	4/6/19
Refine Obstacle Avoidance	Karen & Joseph	3/31/19	4/6/19
Refine Model	Manini	4/6/19	4/13/19
Refine Movement	Karen & Joseph	4/6/19	4/13/19
Refine Model	Manini	4/6/19	4/13/19
Integrate Movement & UI	All	4/13/19	4/20/19
Test Movement	All	4/20/19	4/27/19
Integrate AI & UI	All	27-Apr	4-May
Test	All	27-Apr	4-May

Figure 6: Detailed Schedule of Tasks

B. Team Member Responsibilities

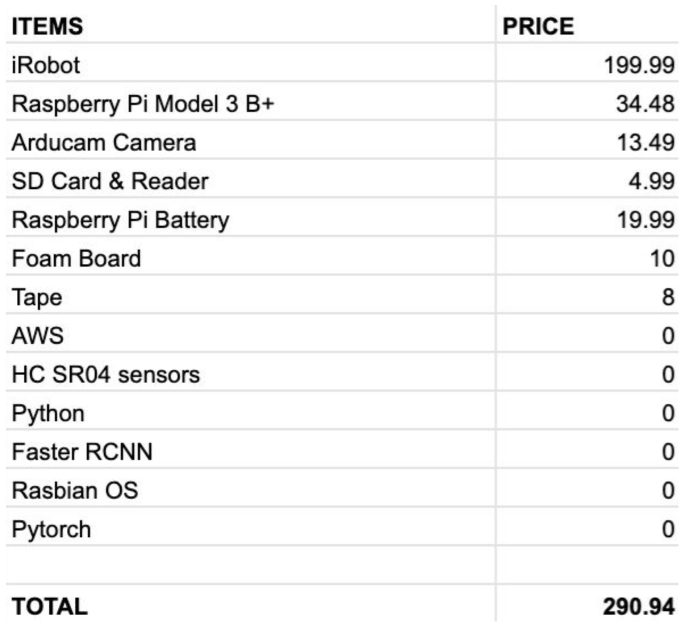
Our responsibilities were split based on the skills that each team member possessed. Karen has extensive hardware experience so she primarily worked with the iRobot Create 2 and the Raspberry Pi. She made sure that the Pi and the Create 2 were able to efficiently communicate with each other and that the Pi was capable of controlling the Create 2. Karen also set up the initial movement commands as well as the path planning algorithm for the Create 2. Finally, Karen's secondary responsibility was designing and building the obstacle course.

Manini has experience with machine learning so she was primarily responsible for the deep learning algorithms of this project. Particularly, she was in charge of the training, biasing, and fine tuning the existing Faster-RCNN architecture so that we can get the desired specifications. Finally, Manini also wrote the pipeline script that transferred the images from the local computer to the AWS instance in the cloud.

Joseph also has experience with machine learning so he started by assisting Manini with setting up the model and with debugging. Afterwards, he worked on obstacle avoidance and integrated that with the path planning and initial movement commands that Karen set up. He also created the UI for the robot and worked with both Karen and Manini to integrate everything together. Finally, he assisted Karen in designing and building the obstacle course.

C. Budget

The majority of our budget was spent on the robotics portion of the project. The Create2 was the most expensive piece, around $200, and all other component prices have been detailed in Figure 7- outlining the costs associated with the bot. An important note is that the entirety of the work done on Amazon Web Services was covered by the free credits that each member was allocated. Thus the detection algorithm and all parts connected to it were essentially free and did not cut into our budget. We were also able to access free breadboards and ultrasonic sensors through various resources around campus. All other free softwares, IDE and languages that were used are also listed below.



ITEMS	PRICE
iRobot	199.99
Raspberry Pi Model 3 B+	34.48
Arducam Camera	13.49
SD Card & Reader	4.99
Raspberry Pi Battery	19.99
Foam Board	10
Tape	8
AWS	0
HC SR04 sensors	0
Python	0
Faster RCNN	0
Rasbian OS	0
Pytorch	0
TOTAL	**290.94**

Figure 7. Budget

AWS Credit Allocation

We used approximately $300 of AWS credits for this project. This included 30 instance hours on a p2.xlarge instance for research and initial testing purposes. Another 35 hours on a p2.8xlarge were used for distributed model training across 8 GPUS and for model inference during our demo and integration tests. These AWS credits were greatly appreciated as our project would not have been possible without the distributed training ML resources the AWS P2 instances provide.

D. Risk Management

We planned on handling our project risk by planning for complexity and anticipating challenges with integration amongst the hardware and software components of our project. We also scheduled multiple slack weeks to account for any unforeseen issues such as the movement library of iRobot being inaccurate.

One of the biggest ways we mitigated risk was by choosing to use an iRobot Create 2 rather than a drone for our project. Drones have a high chance of crashing when piloted incorrectly and since we are trying to achieve a form of autonomous movement, there would most likely be a lot of crashes. This could lead to potential damage on the drone and thus increase our budget significantly. Furthermore, drones require a lot of batteries to use as they have a relatively short flight time. The iRobot, on the other hand, moves relatively slowly so crashing is less concerning. Teams in the past have also been successful with using the iRobot Create 2 so this is a relatively safe solution to our movement problems.

Since we are manipulating the movement of the Roomba using mathematical calculations, we also anticipated error accumulation and therefore planned for extensive testing of basic movement. This was extremely useful as we found through the testing that the iRobot's built-in package for movement was extremely inaccurate and thus we wrote our own code for movement.

Another major risk we had was the possibility that the Faster RCNN model would not work properly. Our risk mitigation plan was to pivot to the YOLO deep learning model if at any point we decide that Faster RCNN no longer suits our project. We extensively researched both of these models and we believed that Faster RCNN was best suited for the project at hand. However, if anything went ary were were prepared to pivot to YOLO as it is a model that we can always work with instead of Faster RCNN.

We also anticipated that training the model would take a good amount of time and therefore started early in obtaining access to Amazon Web Services instances and credits. We significantly limited the classes within the COCO dataset we trained our model on since we are classifying humans instead of general objects to greatly reduce training time.

Finally, we were able to shift different tasks around when certain things were not completed in time. This was especially true when we both experienced a shipping delay and when we found out that the iRobot movement library was not very accurate. During these times we were able to shift the user interface development up to ensure that time was not wasted. This shift inadvertently proved to be another risk management move as it allowed us to approach integration much earlier.

Finally, we dedicated multiple weeks to end to end testing and integration. This time was necessary because combining the various components of our project was surprisingly difficult as we had communication errors between the user interface and the rest of the programs. We were able to overcome this due to the shift in the schedule mentioned above as once we were able to integrate movement with the user interface early.

I. Related Work

The use of drones or robots for search and rescue is not a foreign one. Theoretically any drone with a camera could be sent out to a disaster area to survey for victims. One thing that makes our robot slightly more unique than the typical drone with a camera is that it uses a coordinate system to allow users to determine where to go and then autonomously travels to that location while avoiding obstacles. Another unique feature of our system is how it sends images that the robot takes to the cloud for people detection. Similar robots in this area would be the MIT drone that is capable of navigating a hiking trail and avoiding trees [2] and the work done on creating a swarm of flying drones that are equipped with cameras and thermal sensors to help first responders perform search and rescue [3].

II. Summary

Overall, the SOS_bot was able to successfully complete most of the design specifications. In terms of the ground movement we were able to traverse the grid and arrive at the point of interest with accuracy of 0.5 ft from the specified point. The most optimal path is always calculated by preprocessing the data to find the shortest distance. There is 100% avoidance of obstacles and a distance of 0.5 ft is always maintained throughout the avoidance process. For the human detection portion, we were able to achieve a map score of 53%, which is lower than the original specification of 55%. While this metric was not completely met, we were able to observe high accuracy during the testing and demo phases. We were also able meet and go beyond the specification for the inference time per image on the cloud GPU, lowering from 1 second to 0.8 seconds.

While all the requirements were met, there is always room for growth. If given more time, there are few changes that could be made to the system. Right now, when the bot is taken out of a enclosed space, the ultrasonic sensors are affected by the noise in the environment and detect “obstacles” even when this might not be the case. For better performance, the ultrasonic sensors could be replaced with higher quality sensors or even a Lidar sensor. Additionally, if there was more time, we would be able to widen our scope, allowing for obstacles of larger size. This would require that when an obstacle is detected the bot would retry after a set distance. If the obstacle was of larger size it would have to keep retrying, until the obstacle was cleared. Another way to approach this problem would be to use computer vision to detect the obstacles and estimate how large they are and set the appropriate avoidance procedures. Finally, we could also include a better PID controller or use computer vision to achieve better accuracy of reaching the points of interest.

For a better user experience, it would also be of interest to indicate where on the grid an obstacle was detected and send that image to the user interface as the robot is traversing the grid. It would also be better if the images showed which way the robot was pointing when it took that picture so that first responders could have a better picture of the disaster area. Finally, a system to track where the robot is and give live camera view from the ribot would greatly improve the usability of the robot as first responders would know exactly where the robot is and what it is seeing.

A. Lessons Learned

To future students, we would recommend using a different base for movement. The iRobot Create was overall a very difficult piece of technology to work with. As the robot was run over time, it would respond to commands with less accuracy. This made testing over a prolonged period of time difficult since performance in later hours could not be trusted. Additionally, the library associated with the Create was not a reliable source, and the majority of the functions had to be re-written. We would also recommend students research into better localization methods so that the bot's location can always be tracked in relation to the grid. This will enable an easier process with obstacle avoidance and accuracy.

References

[1] Medium., https://medium.com/@jonathan_hui/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359.

[2] https://www.theverge.com/2016/2/11/10965414/autonomous-drones-deep-learning-navigation-mapping

[3] https://gcn.com/articles/2019/04/12/ai-drone-search-rescue.aspx

[4] https://github.com/jwyang/faster-rcnn.pytorch

[5] https://medium.com/@hichamtout/enabling-bias-in-machine-learning-22cec5d3b820