IntelliSAR

Derek Sun
College of Engineering
University of Massachusetts, Amherst
Amherst, USA
dereksun@umass.edu

Yong Li
College of Engineering
University of Massachusetts, Amherst
Amherst, USA
yonli@umass.edu

Arthur Zhu
College of Engineering
University of Massachusetts, Amherst
Amherst, USA
tianyezhu@umass.edu

***Abstract*—Rescue teams face many challenges when traversing unknown environments. This can be dangerous for both the rescuer and the rescue, so fully understanding the situation is crucial in preventing unnecessary risks. The goal of our project is to provide rescue teams with the ability to remotely examine the situation and the environment to reduce possible risks or dangers and increase the rescue team's efficiency. This paper introduces IntelliSAR, a ground-based robot with artificial intelligence capabilities aimed at supporting post-disaster search and rescue operations. Our system leverages machine learning and computer vision to perform object detection and assist in the task of identifying and locating victims. IntelliSAR supports semi-autonomous navigation as well as manual, remote navigation. The robot's night vision camera and sturdy chassis allow it to remain effective in low-light and rugged environments. In addition, the robot is equipped with a temperature and humidity sensor and an ultrasonic sensor. The data from all the sensors are wirelessly transmitted back to an external PC via Wi-Fi and serve to provide the user with a wide range of real-time environmental information.**

Keywords—IntelliSAR, search and rescue, object detection, semi-autonomous navigation

I. Introduction

Humans have been fighting against natural disasters for thousands of years. Cave landslides, flash floods, mudslides, and earthquakes are just a few examples of impactful and dangerous natural disasters. The number of earthquakes with a death toll in the 21st century will increase, and the number of people killed by earthquakes will exceed that of the past. "Combining fatalities caused by the background rate with fatalities caused by catastrophic earthquakes (>100,000 fatalities) indicates global fatalities in the 21st century will be 2.57±0.64 million if the average post-1900 death toll for catastrophic earthquakes (193,000) is assumed" [1].

When faced with these dangerous natural disasters, it is important for search and rescue tasks to be carried out as efficiently as possible. The first step in starting a rescue after a natural disaster is to search for victims [2]. Searching for victims in a post-disaster situation requires accuracy, speed, and flexibility. Rescuers must be able to get reliable information on the post-disaster situation through environmental observation and testing. Information lays the foundation for correct and efficient implementation of subsequent rescue work. Rescue workers, materials, and rescue facilities must be able to quickly arrive and set up following a disaster. Rescuers can then begin the search and rescue work as soon as possible, thereby gaining valuable time, improving rescue efficiency, and helping more victims survive. Flexibility is necessary due to the unpredictable situations and harsh environments that may be present in a post-disaster situation.

In order to accommodate the aforementioned characteristics, our team developed IntelliSAR; an intelligent, ground-based robot designed to be used in post-disaster search and rescue situations. Search and rescue focus on locating and extracting people trapped in a collapsed or damaged structure. In these types of situations, rescuers are under extreme time pressure; after 48 hours, the mortality rate drastically increases due to lack of air, food, water, and medical treatment. Attempting to rescue victims can be as dangerous as the initial disaster for both the victim and rescuer [3]. In such conditions, lightweight and intelligent robots can greatly benefit search and rescue initiatives by exploring ahead of rescue teams and reporting conditions that may be hazardous [4]. IntelliSAR was designed with this goal of supporting rescue teams in mind. Through a user-friendly web application, the operator can remotely control IntelliSAR, view live temperature and humidity data, and view a live, night vision enabled video feed. Additionally, IntelliSAR is able to perform object detection and assist in the task of identifying and locating victims through the use of machine learning and computer vision techniques. All this information is relayed to the operator through the web application and allows rescue teams to thoroughly examine the post-disaster situation without exposing themselves to potential dangers.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 discusses the details and specifications of IntelliSAR's design. Section 4 discusses implementation, results, and analysis of our prototype. Section 5 describes and shows how the IntelliSAR system would have looked had we completed it. Section 6 discusses the project management aspects and our team dynamic. Section 7 concludes the paper with a summary of our prototype and project results.

II. Related Work

Search and rescue robots provide numerous benefits for disaster response and have been used following several high-profile disasters such as 9/11 and Hurricane Katrina [5]. These robots are constantly being developed and improved upon in order to better enhance search and rescue efforts.

A few examples of modern search and rescue robots are the Inuktun VGTV-Xtreme [6], iRap Robot [7], and iRobot PackBot [8].

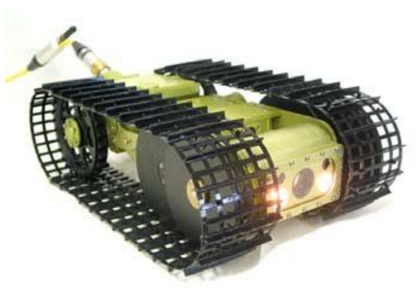
Figure 1. Inuktun VGTV-Xtreme [6]

The Inuktun VGTV-Xtreme, shown in figure 1, was originally designed for industrial inspection purposes, but was adapted to fit a search and rescue role. This robot was first used in 2001 during the World Trade Center disaster and went through several design improvements following that. The main features of the Inuktun VGTV-Xtreme are the compact size, remote video feed, and considerable maneuverability provided by the tracked design. However, it was confirmed to have a short battery life of less than ten minutes during the 2005 La Conchita Mudslides. The Inuktun VGTV-Xtreme was not fitted with any semi-autonomous or autonomous driving capabilities because of its early design. Despite its relatively straightforward and simple design, the Inuktun VGTV-Xtreme was one of the first explorations into using robots for search and rescue purposes and effectively fulfilled the ultimate purpose of providing responders with more information about the environment.

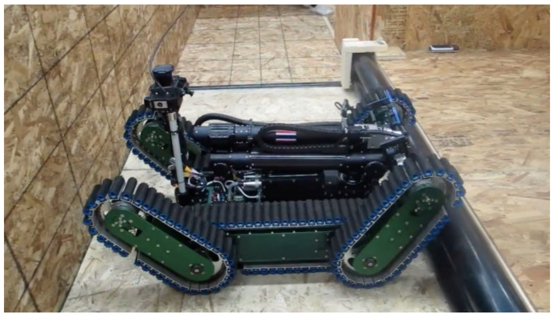
Figure 2. iRap Robot [7]

The iRap Robot, shown in figure 2, was designed and created for the Robocup Rescue 2018 competition. This robot features remote exploration, motion detection, 2D map generation, thermal imaging, and high maneuverability. The iRap Robot's dimensions are 100x60x60 cm, but it can reach 100x60x200 cm when standing up. This robot was reported to have cost approximately 30,000 USD to create, with the bulk of it spent on the 6-axis robot arm [7]. The iRap Robot's remote exploration and object/motion detection features are similar to IntelliSAR, but the iRap Robot focuses more on competitive maneuverability and robustness. Some of the drawbacks of their approach are the high cost and the large and bulky chassis.

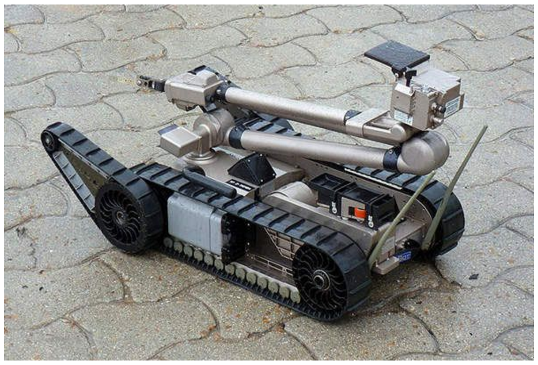
Figure 3. iRobot PackBot [8]

The iRobot PackBot, shown in figure 3, was designed for military personnel in high-threat battlefield scenarios and can be used for surveillance and reconnaissance, bomb disposal, vehicle inspection, and various other dangerous missions. The iRobot PackBot is remotely controlled with a few semi-autonomous features and has dimensions of 70x50x20 cm. Purchasing an iRobot PackBot costs about 100,000 to 200,000 USD [9]. The iRobot PackBot's search-and-rescue features are similar to IntelliSAR's, but the PackBot is more focused on military utility and robustness and reliability of the robot. The main drawback is the extremely high cost for each unit.

Compared with other modern search and rescue robots, IntelliSAR's biggest advantage is its low cost. IntelliSAR is highly practical and cost-effective. The main service of IntelliSAR is search and rescue. IntelliSAR is suitable for this role with its accurate object detection and robustness and was not designed with military use in mind. The use of an ultrasonic sensor instead of a Lidar sensor for autonomous navigation helps save costs without drastically reducing the reliability of the autonomous navigation. Table 1 shows a comparison between IntelliSAR and the other three aforementioned search and rescue robots.

	IntelliSAR	VGTV-Xtreme	iRap Robot	iRobot PackBot
Size	Small	Medium	Large	Medium
Communication	WiFi	Radio	WiFi/ Radio	Radio
Object Detection	Person	N/A	Hazmat/ QR code	N/A
Navigation	Semi-auto/ Manual	Manual	Auto/ Manual	Semi-auto/ Manual
Navigation Sensor	Ultrasonic	N/A	Lidar	Stereo Camera, Lidar
Target Audience	Search and Rescue	Search and Rescue	Search and Rescue	Military
Cost	Low (<$500)	High (~$10000)	High (~$30000)	Very high ($100000+)

Table 1. Comparison of IntelliSAR and other Search and Rescue Robots

III. Design

This section discusses the details of IntelliSAR's design and implementation.

A. *System Overview*

IntelliSAR consists of the following components: Yahboom G1 robot chassis [10], Raspberry Pi 4B [11], 4WD expansion board [12], Yahboom horizontal ultrasonic distance sensor [13], DHT11 temperature sensor [14], Coral USB accelerator [15], MakerFocus Raspberry Pi camera [16], Yahboom 370 motors [17], SG90 servos [18], and battery pack. Figure 4 shows the built IntelliSAR system with labeled components.

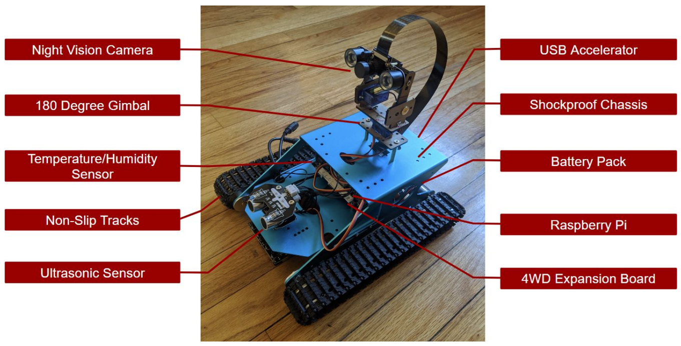

Figure 4. IntelliSAR with Labeled Components

The aluminum alloy chassis securely houses the individual components in their designated locations. The Raspberry Pi and 4WD expansion board are both mounted beneath the upper platform on the lower platform. The ultrasonic sensor is mounted on a servo near the front of the robot. The temperature sensor is mounted towards the middle of the chassis, near the 4WD expansion board. The USB accelerator is mounted at the back of the chassis, beneath the upper platform. The camera is attached to a gimbal system made up of two servos on the upper platform of the chassis. The robot's treads are rotated by two motors; one on each side at the rear of the chassis. The battery pack is mounted on the bottom side of the chassis.

Figure 5 shows the data flow throughout IntelliSAR's overall system.

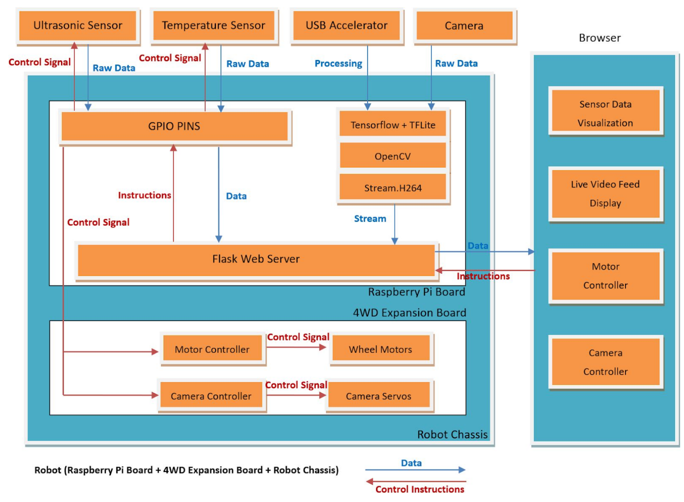

Figure 5. Block and Data Flow Diagram

The on-board Raspberry Pi 4B is the brain of the system and is responsible for hosting the web server, interfacing with the motor controller and camera controller on the 4WD expansion board, processing all the sensor data, and performing object detection. The Flask web server allows a remote operator to send control instructions to the robot and view the processed sensor data and live video feed. The robot and the remote browser are connected by Wi-Fi or mobile hotspot, and all data transfer is through the http protocol. The Raspberry Pi sends signals to the motor controller to control the direction and speed of the wheel motors and sends signals to the camera controller to rotate the servos that make up the camera gimbal system. The raw distance data from the ultrasonic sensor is captured and calculated by the Raspberry Pi for use in the semi-autonomous navigation feature. The raw temperature data is processed and converted into a readable format by the Raspberry Pi so that it can be displayed on the web application. The camera sends the raw image data to the Raspberry Pi via the camera bus. The image data is first processed by the object detection component of the application, and then the output H.264 video feed with the resulting detections is sent to the web server. The USB accelerator is connected to the Raspberry Pi's USB port and helps with the processing and computations needed for the object detection.

Table 2 shows the specifications of the IntelliSAR system.

Specification	Value
Weight	7 lb
Dimensions	256 x 183 x 213 mm
Battery	3.7v 18650 Battery x 3
Battery Life	~120 min
Control Distance	50 m indoors, 100 m outdoors
Camera	Night vision, 5MP, 1080p
Temperature Measurement Range	0 ~ 50 ℃
Temperature Measurement Accuracy	± 2 ℃
Speed Range	0.7 ~ 6.5 km/h
Camera Rotation	Horizontal: 0°~180° Vertical: 45°~180°
Obstacle Detection Range	2 ~ 500 cm
Obstacle Detection Accuracy	0.3 cm
Object Detection Range	6 m (best case scenario)
Video Stream w/ Object Detection Frame Rate	H.264 640x480 @ 30FPS

Table 2. Specifications for IntelliSAR

A weight of 7 lb and dimensions of 256x183x213mm allow IntelliSAR to maintain a small form factor and convenient setup/deployment. IntelliSAR has a large enough battery to guarantee two hours of normal operation (analyzed in section IV, part D). In terms of remote control range, IntelliSAR is able to be controlled from up to 50 meters away indoors or up to 100 meters away outdoors (analyzed in section IV, part E). The control range is much lower indoors because the radio frequency cannot penetrate solid objects such as walls and floors. The camera supports night vision and a max resolution of 1920x1080, but we chose to use 640x480 to lower the amount of processing power needed for the object detection. The DHT11's temperature measurement range is from 0 to 50 degrees Celsius, with an accuracy of ± 2 degrees Celsius [14]. The speed at which IntelliSAR moves is adjustable, with a minimum speed of 0.7 km/h and a maximum speed of 6.5 km/h. With the ultrasonic distance sensor, IntelliSAR is able to detect obstacles up to 5 meters away with an accuracy of 0.3 centimeters [13]. In terms of object detection range, IntelliSAR is able to detect people/victims up to 6 meters away in the best case scenario of non-blurry image and clearly distinguishable human form. The object detection enabled video stream shown on the web application is 640x480 resolution and runs at approximately 30 FPS. Our specification analysis helps us to better evaluate the performance of IntelliSAR and determine areas in need of optimization. IntelliSAR meets all of our specification goals, but we also determined that several important specifications such as battery life, speed, and obstacle detection accuracy could be improved by simply purchasing upgrades to the corresponding hardware components.

B. *Robot*

The on-board Raspberry Pi 4B is responsible for hosting the web server, interfacing with the motor controller and camera controller on the 4WD expansion board, processing all the sensor data, and performing object detection.

The manipulation of the robot includes controlling the wheel motors and the camera platform. Two wheel motors are used to move the robot in different directions at adjustable speeds. The camera is mounted on a platform controlled by two servos in order to provide a greater viewing scope. The user interface is provided through the web page, through which the operator is able to control the robot's moving direction, speed, and camera rotation. The navigation instructions are sent to the Flask web server on the Raspberry Pi as http requests, and these instructions are passed to the GPIO PINs that connect to the motors and servos.

The ultrasonic distance sensor is used for obstacle detection and avoidance in the IntelliSAR system. Every second, the application sets the PIN_TRIGGER voltage from 0 to 1 for 20ms, instructing the sensor to send out an ultrasonic pulse. Once the sensor receives the reflected waves, the sensor sets the PIN_ECHO to 1. The application monitors the PIN_ECHO and records when the reflected wave returns. The distance between the robot and the surrounding obstacles can be calculated as $\frac{Time \times Sound\ Speed\ in\ Air}{2}$. These calculated distances are used to analyze the distance of the obstacles detected and determine the robot's next movement direction. The navigation instructions are sent to the motor controller via the GPIO PINS and are then converted to control signals to drive the wheel motors. Figure 6 shows the ultrasonic sensor and the aforementioned pins.

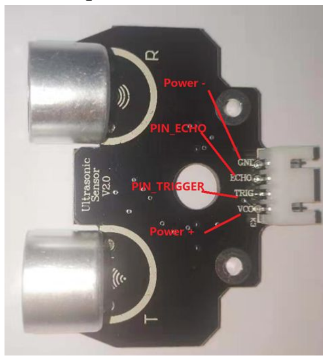

Figure 6. Ultrasonic Sensor

Similarly, the environmental sensors capture the surrounding temperature and humidity data and transfer the data to the Raspberry Pi through the connected GPIO PINs. The Raspberry Pi acquires the raw sensor data and interprets it to output meaningful temperature and humidity values. Once an http client requests the web server for the environmental data, the web server will poll the values and display them on the client.

The Coral USB accelerator, with its on-board Edge TPU (Tensor Processing Unit), is designed to speed up inferencing on edge devices such as IntelliSAR's on-board Raspberry Pi [15]. The Edge TPU is an ASIC (application-specific integrated circuit) chip designed by Google that uses highly parallelized and directly connected ALUs (arithmetic logic units) to achieve a high computational throughput on the calculations necessary in a neural network. Due to the efficient parallelization and removal of memory accesses, the Edge TPU is able to do this with less power consumption and a smaller footprint [19]. It is secured to the back of the robot chassis and connected to the Raspberry Pi via USB cable. The incorporation of the Coral USB Accelerator allows us to drastically increase our object detection video feed's frames per second from 4 to 30 and provide a much better user experience without having to sacrifice any functionality or accuracy. The Coral USB accelerator is shown in figure 7.

Figure 7. Coral USB Accelerator

C. *Web Application*

In order to enable easy deployment and remove limitations such as installing and configuring software for robot operation, we decided to use a web application for displaying the data and controlling the robot. This also allows the robot to be used over a wide range of devices, including laptops, mobile tablets, and phones. We used the Python Flask web framework for IntelliSAR's web application because it is lightweight and easy to work with. When designing the webpage, the Bootstrap CSS framework was used to ensure compatibility with all devices no matter the screen size. Figure 8 shows the user interface that we designed for our web application. The live video feed with toggleable object detection is shown in the center of the UI. The buttons to control the direction and speed of IntelliSAR's wheel motors are located on the left side, and the buttons to control the servos of the camera gimbal system are located on the right. At the very bottom is the button to turn on or off the semi-autonomous navigation. The live temperature data is viewed on a separate page that can be accessed through the dropdown menu at the top right.

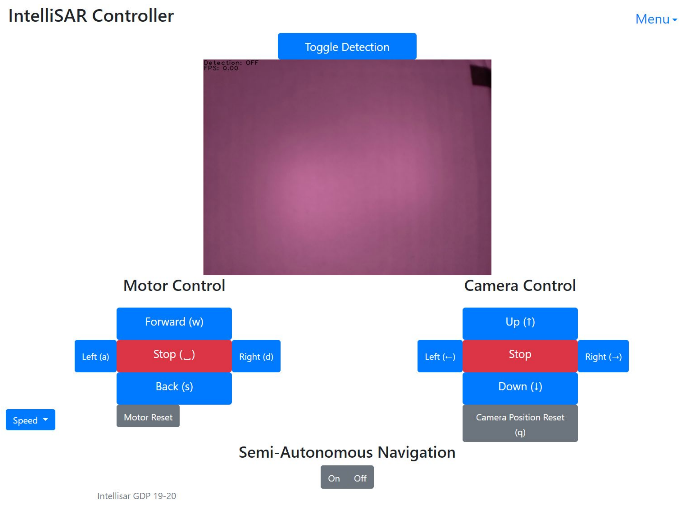

Figure 8. Web Application UI

D. *Semi-Autonomous Navigation*

There is one ultrasonic distance sensor mounted on a servo at the front side of the robot chassis. The sensor sends out ultrasonic waves that get reflected back and captured if an obstacle is encountered. The distance between the sensor and the obstacle is calculated by the following formula:

$$Distance = \frac{(t_{received} - t_{sent})}{2} * V_{sonic}$$

By monitoring the surrounding space, IntelliSAR is able to avoid obstacles and navigate semi-autonomously. The currently implemented algorithm is described by the flowchart shown in figure 9. IntelliSAR continues forward until the distance sensor detects that an obstacle is within 30 centimeters. Then, IntelliSAR turns right, turns left, or reverses backwards, based on the data returned by the pivoting distance sensor. We used a distance of 30 centimeters for our algorithm because that is the minimum distance at which IntelliSAR is able to successfully turn around.

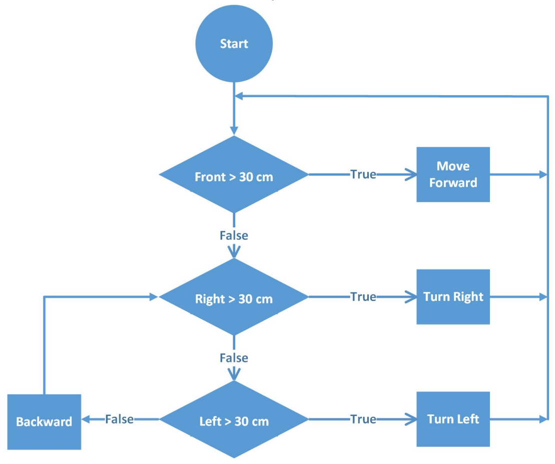

Figure 9. Semi-Autonomous Navigation Flowchart

E. *Object Detection*

In order to better support post-disaster search and rescue operations and help increase the efficiency of rescue teams, IntelliSAR provides person detection capabilities through the use of machine learning and computer vision techniques. This function is able to reliably detect victims in various poses such as standing straight, sitting down, and lying down and was implemented using a custom trained object detection model.

In order to custom train our detection model in an efficient manner, we used a technique called transfer learning. Transfer learning is a process where a neural network model pre-trained in a related domain is used to accelerate the development of models in a custom or more specific domain [20]. The process consists of restoring the weights of the pre-trained model and training the model on one's own custom dataset, thereby fine tuning the layers from the pre-trained model. In the rest of this section, we first introduce commonly used object detection terminology and metrics and then discuss a few of the object detection architectures that we explored, tradeoffs between them, the specific object detection model that we determined would be the best starting point for training our custom model, and the goals that we adhered to during the training process.

1. *Object Detection Metrics*

This section will discuss and define terminology and metrics that are commonly used to measure the performance of object detection models [21]. Precision measures the model's prediction accuracy and is defined as:

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} = \frac{True\ Positives}{Total\ \#\ of\ Predicted\ Positives}$$

Recall measures how well the model finds all the positives and is defined as:

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

In the case of IntelliSAR's person detection:

$$Precision = \frac{\#\ People\ correctly\ predicted}{\#\ People\ correctly\ predicted + \#\ Objects\ incorrectly\ predicted\ as\ person}$$

$$Recall = \frac{\#\ People\ correctly\ predicted}{\#\ People\ correctly\ predicted + \#\ People\ incorrectly\ labeled\ as\ non\text{-}person}$$

Intersection over Union (IoU) measures the overlap between the bounding box generated by the model and the ground truth bounding box and is what determines whether a prediction is a true positive, false positive, or false negative.
IoU is defined as:

$$IoU = \frac{Area\ of\ overlap\ for\ predicted\ box\ and\ ground\ truth\ box}{Area\ of\ union\ for\ predicted\ box\ and\ ground\ truth\ box}$$

For example, if a metric is using IoU=50%, then a prediction would only be marked as "correct" if the predicted bounding box has a 50% or more overlap with the corresponding ground truth box.

Average precision (AP) is defined as the area under the precision-recall curve (PR curve), with the recall on the x-axis and precision on the y-axis.

Mean average precision (mAP) is calculated by taking the average of the AP for all the classes being predicted. For a person detection system, the mAP would be the same as the AP because only the single Person class is being predicted.

2. *Faster Region Convolutional Neural Network (R-CNN)*

The Faster R-CNN architecture, shown in figure 10, performs detection in two stages: proposal generation and box classification [22]. The proposal generation stage, called the region proposal network (RPN), consists of processing input images using a feature extractor and then predicting class-agnostic box proposals based on features at some selected intermediate level. The box classification stage consists of using these box proposals to crop features from the same intermediate feature map and then feeding the cropped features to the feature extractor to predict a class and class-specific box refinement for each proposal. The Faster R-CNN architecture has been particularly influential, with its concepts becoming the basis for many other detection architectures (e.g. R-FCN and SSD).

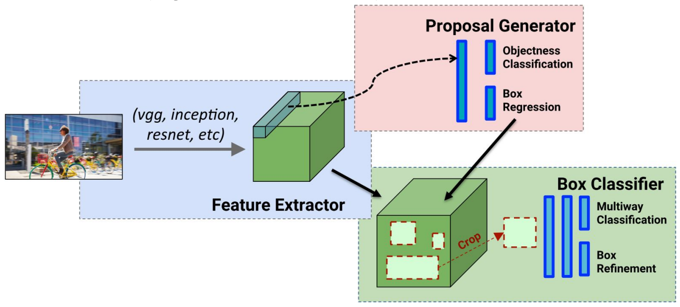

Figure 10. Faster R-CNN Architecture [22]

3. *Region-based Fully Convolutional Network (R-FCN)*

The R-FCN architecture, shown in figure 11, is similar to Faster R-CNN, but handles the feature cropping in the second stage more efficiently. Crops are taken from the last layer of features prior to prediction, rather than from the same layer where region proposals are predicted [22]. This approach minimizes the amount of per-region computation necessary and allows R-FCN to achieve comparable accuracy to Faster R-CNN, often at a faster speed.

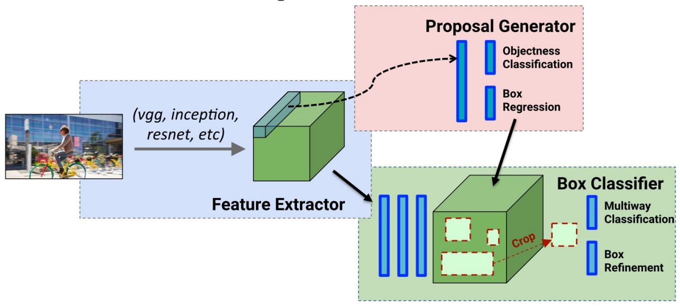

Figure 11. R-FCN Architecture [22]

4. *Single Shot Detector (SSD)*

The Single Shot Detector (SSD) architecture, shown in figure 12, typically uses a single feed-forward convolutional network to directly predict classes and anchor offsets without requiring a second stage per-proposal classification operation

[22]. When related to Faster R-CNN, SSD architectures modify the proposal generation stage to also directly output class probability and anchor offsets. This allows SSDs to perform detection in a “single shot”.

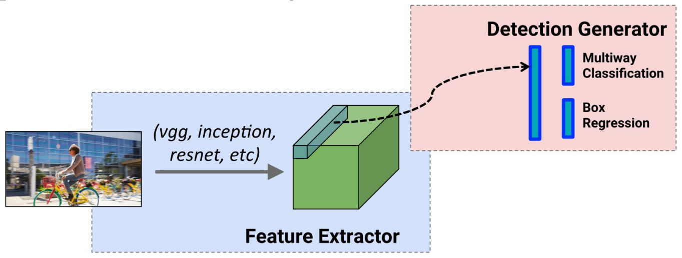

Figure 12. SSD Architecture [22]

5. *Speed/Accuracy Tradeoffs*

To select the right detection architecture for our specific use case, we considered each architecture’s attributes, namely speed and accuracy. Speed is important because we are running the object detection on a Raspberry Pi but still want to achieve an acceptable frame rate. Accuracy is important because IntelliSAR needs to be able to reliably detect victims. In the rest of this section, we will discuss the results of Jonathan Huang et al.’s speed and accuracy investigation on three common, state-of-the-art detection architectures [22].

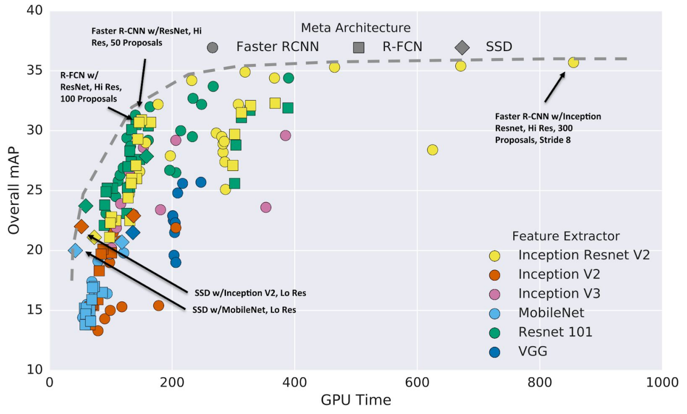

Figure 13. Accuracy vs Time [22]

Figure 13 shows the results of the authors’ accuracy vs time testing. In this scatterplot, the x-axis represents the GPU time, and the y-axis represents the mean average precision. GPU time is the measurement of timings on their GPU in milliseconds, with the results ranging from tens of milliseconds to almost one second. Mean average precision is a measure of the model’s percentage of correct predictions for all classes. The point’s shape indicates the architecture and the point’s color indicates the feature extractor used. Each architecture and feature extractor pair may correspond to multiple points due to the changing of other parameters (e.g. number of proposals). The points of interest are marked in the scatter plot. Shown on the upper right, Faster R-CNN paired with the Inception Resnet feature extractor has very high accuracy with very slow speed. Faster R-CNN architectures typically have high accuracy and low speed, but they can be significantly sped up with only a small decrease in accuracy by reducing the number of proposals used. Shown on the upper left, the same setup using 50 proposals instead of 300 proposals runs significantly faster with only a minimal reduction in accuracy. R-FCN, as mentioned previously, is able to achieve comparable accuracy to Faster R-CNN with much faster speeds. R-FCN with Resnet and 100 proposals boasts performance similar to Faster R-CNN with Resnet and 50 proposals and also runs significantly faster with only a minimal reduction in accuracy when compared to Faster R-CNN with 300 proposals. SSDs are much faster than other detection architectures but have decreased accuracy, especially on small or tightly clustered objects. As shown in the scatterplot, SSD with InceptionV2 and SSD with MobileNet run extremely fast but have lower accuracy than Faster R-CNN and R-FCN. In the end, each detection model has their own performance tradeoffs that need to be considered for one’s use case. The choice of which specific object detection model would be best suited for the implementation of IntelliSAR was made based on the aforementioned speed and accuracy investigation and is discussed in detail in the next section.

6. *Basis for our Custom Model*

For our project, we used a quantized SSD MobileNetV2 model pre-trained on the Common Objects in Context (COCO) dataset as the basis for our detection model. This model was obtained from Tensorflow’s object detection model zoo [23], a repository that provides developers with several pre-trained detection models of various different neural network architectures. The quantized version of the SSD MobileNetV2 reduces both the memory requirement and computational cost of the original model. We chose this model because it was designed for mobile and edge devices and suits our requirements of performing real-time object detection directly on IntelliSAR’s on-board Raspberry Pi. Based on the results shown in figure 13, the SSD with MobileNet model runs approximately 17 times faster than Faster R-CNN with ResNet and 300 proposals. The SSD with MobileNet model also runs approximately 3 times faster than both R-FCN with ResNet and Faster R-CNN with ResNet and 50 proposals. It is important to note that these are all relative timings and may vary depending on the hardware used, but they still provide valuable insight into the performance of each model. Additionally, these comparisons do not take into account the effects of optimizations such as quantizing the model or converting the model into a Tensorflow Lite model, both of which are currently only able to be applied to SSD models in Tensorflow. The SSD detection architecture has relatively low accuracy when compared to architectures such as Faster R-CNN or R-FCN, but we deemed that SSD’s drastic speed improvement was worth the 0.10 to 0.15 decrease in mean average precision. The quantized SSD MobileNetV2 model pre-trained on the COCO dataset from Tensorflow’s model zoo is able to detect objects of 80 different classes and has a mean average precision of 0.22 based on COCO's evaluation metric, serving as a solid basis for creating our custom detection model.

7. Training Goals

In order to ensure that the model is training properly and is able to achieve acceptable performance and accuracy, it is important to evaluate the model constantly, avoid underfitting, and avoid too much overfitting. Underfitting and overfitting are both significant issues that may occur while training a machine learning model, with overfitting being the more prevalent issue of the two. Underfitting occurs when the model does not fit the training data and misses the trends or patterns in the data. Underfitting can typically be solved by just training the model for a longer amount of time. Overfitting occurs when the model has trained "too well" and is fit too closely to the training dataset, meaning that it doesn't generalize well on new data [24]. Having a bit of overfitting is actually beneficial for your model's performance, but letting it overfit too much can quickly ruin a model's performance. One method of avoiding too much overfitting is to use a train/test split, meaning that the dataset is split into a training set and an evaluation set. The training set is used to train the model, and the evaluation set is used to evaluate the model periodically over the training process. This method allows one to check the model's ability to generalize on unseen data (from the evaluation set) and stop the training when the model begins to overfit. Another method of avoiding overfitting is to use more complete training data that covers the full range of inputs that the model is expected to handle. With a perfect dataset, it would be beneficial for the model to fit as closely as possible. However, obtaining a perfect dataset is impossible, unless the detection model being trained is expected to only handle a very small range of inputs. Overall, the main goals during the training of our model were to use as comprehensive a dataset as we could obtain, get the evaluation error as low as possible (or evaluation mAP as high as possible), get the evaluation error similar to or slightly higher than training error, avoid underfitting, and avoid too much overfitting.

IV. Prototype Implementation

In this section, we will discuss the implementation, results, and analysis of our prototype up through the Comprehensive Design Review (CDR). Table 3 shows the status of the previous Midway Design Review (MDR) deliverables that we proposed during Preliminary Design Review (PDR). Table 4 shows the status of the CDR deliverables that we proposed during MDR.

MDR Deliverables	Status
Functional robot able to be remote controlled	**Complete** Robot is remote controllable through a Python Flask web server. The night vision and object detection enabled live video feed, and temperature sensor data are viewable through the webserver.
Azure setup for our system	**Design Change** Following feedback from PDR, we decided to detach from the cloud and only transmit data over WLAN.
Trained object detection model able to detect and classify certain objects	**Complete** Custom-trained a SSD MobileNetV2 model that is able to detect people in various positions. A Python script that is run on the Raspberry Pi streams the live video feed, with bounding boxes and classifications shown, to a Flask web server.

Table 3. Status of MDR Deliverables

CDR Deliverables	Status
Reconstructed robot with all functionality restored	**Complete** Original robot is inaccessible, so IntelliSAR has been reconstructed with newly purchased robot parts. All software functionality has been fully restored.
Improve accuracy of object detection	**In Progress** Overall mAP increased from 0.2255 to 0.3554.
Improve speed of object detection	**Complete** Purchased a Coral USB accelerator and integrated it into our system. Frames per second of object detection video feed increased from 4 fps to 30 fps.
Re-implement semi-autonomous navigation and improve reliability	**In Progress** Semi-autonomous navigation functionality was restored in the reconstructed robot. Increased turn speed and stopping reliability.
Trained object detection model able to detect and classify multiple objects	**Complete** Object detection model trained to detect both people and rocks.

Table 4. Status of CDR Deliverables

A. *Robot Functionality*

Regarding the robot deliverable, the mobility and manipulation of the robot has been implemented. Two motors are used to drive the robot's treads on each side and are controlled through the user interface on the Flask web server. This web server is hosted and handled by the on-board Raspberry Pi. On the control page, the operator is able to control the robot via the clickable buttons or the keyboard controls and make it move forward, move backward, turn right, turn left, or stop. The robot's movement speed is controlled through the pulse width modulation (PWM) value on the GPIO PINs that the motor driving board is connected to. By adjusting the PWM value to the left and right motors, different amounts of power are delivered to the motors. When the motors rotate at full speed, the robot moves at a speed of 6 km/hour. To make the robot easier to control and avoid potential accidents, we set the default speed to be 40% of the full speed. Through the web UI, the operator is able to adjust the speed to anywhere between 0% and 100% of the full speed.

The live video feed is provided by the camera on the robot, which is fixed on top of a rotatable platform. Two servos are used to rotate the camera platform horizontally and vertically. There are many choices for video encoding, but we primarily considered mjpeg and H.264. The mjpeg encoding sends each frame as a jpeg formatted picture and thus requires more network bandwidth and increases latency. We chose to use the H.264 encoding because it is more efficient for transferring real time video over a wireless network.

Similar to the robot controller, the environmental sensor data is hosted on another webpage. The temperature data and humidity data from the sensors on the robot are passed to the Raspberry Pi and displayed on the web server. This data is rendered as two dynamic curves, with the values and times also displayed at the bottom of the page.

The semi-autonomous navigation is initiated by an on/off button on the web UI. Once the button is clicked, IntelliSAR begins polling the ultrasonic sensor to monitor the surrounding space and traverse while avoiding any obstacles. When the semi-autonomous navigation is turned off, IntelliSAR halts its current movement and waits for control commands from the operator.

Due to customs issues causing an inability to continue working with the original robot constructed in the first project semester, we purchased new robot parts and reconstructed IntelliSAR. However, our existing software was not fully compatible with the new robot platform and had to be updated accordingly. All software functionality was fully restored for CDR.

B. *Object Detection*

The object detection functionality is implemented through the use of the Python programming language and the Tensorflow, Tensorflow Lite, and OpenCV software libraries. Tensorflow is an open-source machine learning library. Tensorflow Lite is an open-source deep learning framework specifically designed for edge computing applications and helps developers run Tensorflow models on their mobile, embedded, and IoT devices. OpenCV is an open-source computer vision library that provides many useful functions for object detection applications. Tensorflow's Object Detection API [25], an open source framework that makes it easier for developers to construct, train and deploy object detection models, was used to train and evaluate our custom detection model and convert the model into an optimized Tensorflow Lite model. Tensorflow Lite models have a faster inferencing time and require less processing power, thus running at higher speeds than regular Tensorflow models. The Tensorflow Lite interpreter was used to run our converted, custom-trained Tensorflow model on IntelliSAR's on-board Raspberry Pi 4B. The Coral USB accelerator, with its on-board Edge TPU, was purchased and integrated into our system prior to the Comprehensive Design Review. We used Google's Edge TPU Compiler [26] to compile our custom Tensorflow Lite model for use with the Edge TPU. The addition of the Coral USB accelerator and its Edge TPU allowed for an increase in the frames per second of the object detection video feed (from 4 fps to 30 fps).

Initially, we used Google's Open Images Dataset v5 (OIDv5) [27] to provide the labeled images needed for training our custom model. The OIDv4 ToolKit [28] was used to obtain only images and annotations of the "Person" class, allowing us to avoid downloading the whole dataset (~9 million images). For the training of the model, we used approximately 6250 images. The model trained with this dataset was showcased at the Midway Design Review.

To improve the accuracy of IntelliSAR's object detection following the Midway Design Review, we trained the model on our own dataset of images that better covers the range of inputs that the model is actually expected to handle. In IntelliSAR's case, we compiled an image database of people in different positions, in different lighting, and with varying amounts of noise. These images were all taken with the Raspberry Pi's camera in both night vision on and night vision off modes. The labeling of the images was done manually via the graphical image annotation tool LabelImg [29]. The dataset for training consisted of 250 labeled images of people. The dataset used for evaluation consisted of 70 labeled images of people.

Additionally, we trained our model on labeled images of rocks to demonstrate that our object detection system is able to detect and classify multiple objects at a time. To accomplish this, we took pictures of natural rocks of varying sizes and shapes and used LabelImg to label them. In the end, we added 30 labeled images of rocks to the training dataset and 10 labeled images of rocks to the evaluation dataset.

The object detection enabled video feed was implemented with Python and has been integrated with the main IntelliSAR web application. When the Flask web application is started, the program starts up the 640x480 videostream and loads our custom detection model into the Tensorflow Lite interpreter. Then, each frame of the videostream is processed using the Tensorflow Lite interpreter, and we get back the predictions in the form of bounding box locations, classifications, and confidence. Finally, we use OpenCV to draw these predictions on the frame and display it on the web application's video feed. We also calculate the framerate of our object detection video feed using OpenCV's tick functions and display it in the top left corner of the frame.

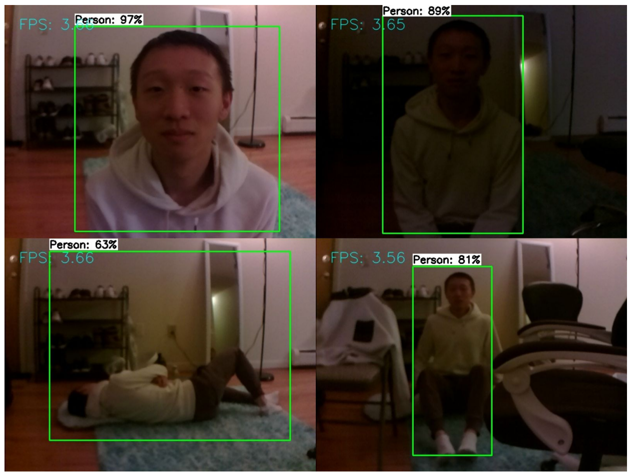

Figure 14. Examples of Object Detection

In figure 14, we show examples of detecting a close-up person, a close-up person in dim lighting, a person lying down on the floor, and a person surrounded by several random objects. Detecting a close-up person is very reliable due to the large size and well-defined features. Detecting a close-up person in dim lighting is slightly less reliable due to the lower visibility of the features, but still performs well. Detecting a person lying down on the floor is much less reliable due to the smaller size and often complete absence of prominent person-defining features. Detecting a person surrounded by several objects helps show that the object detector is truly detecting only people and not just any random object in the frame.

C. *Object Detection Analysis*

This section discusses numerical evaluation metrics (defined in section III, part E.1) for our object detection model.

Metric	Value
mAP	0.2255
mAP (large)	0.2782
mAP (medium)	0.04057
mAP (small)	0.0016068
mAP@.50IOU	0.4332
mAP@.75IOU	0.2044

Table 5. Detection Model Evaluation Metrics (MDR)

Metric	Value
mAP	0.3554
mAP (large)	0.3968
mAP (medium)	0.0505
mAP (small)	0.0022644
mAP@.50IOU	0.6514
mAP@.75IOU	0.375

Table 6. Detection Model Evaluation Metrics (CDR)

Table 5 shows several metrics that were obtained by evaluating the initial custom trained model, which was displayed at the Midway Design Review, on our evaluation dataset. Table 6 shows the same metrics for the updated model displayed at the Comprehensive Design Review. The mAP metric is obtained by averaging the mAPs calculated using IoU thresholds ranging from .5 to .95 with increments of .05 [30]. The mAP (large) metric is the calculated mAP for large objects (96^2 pixels < area < 10000^2 pixels). The mAP (medium) metric is the calculated mAP for medium-sized objects (32^2 pixels < area < 96^2 pixels). The mAP (small) metric is the calculated mAP for small objects (area < 32^2 pixels). The mAP@.50IOU metric is the mAP calculated using an IoU threshold of 50%. The mAP@.75IOU metric is the mAP calculated using an IoU threshold of 75%.

The quantized SSD MobileNetV2 model pre-trained on the Common Objects in Context (COCO) dataset has a mAP of 0.22. However, the mAP of our custom model and the mAP of the pre-trained model are calculated using different evaluation datasets and can't be directly compared. Despite this, the mAP of the pre-trained model still provides a good frame of reference because the mAP of our custom model should become more similar to the mAP of the pre-trained model as we improve the accuracy of our model and make the evaluation dataset more thorough.

D. *Power Consumption Analysis*

The robot's battery pack consists of three 3.7v 18650 batteries. To understand how long the battery can last, we calculated the power consumption based on the specifications of each component. After charging, the battery voltage is about 12.6V. According to the Raspberry Pi specifications [31], the minimum input current has to be 3A when the current draw of the peripherals is higher than 0.5A. Through calculation, we can conclude that the time that the car can work normally is 108 minutes, which is consistent with our experimental data. Table 7 describes the main board power consumption. Table 8 describes the driving board power consumption. Table 9 describes the robot power consumption. Table 10 describes the battery life analysis.

Main Board Power Consumption				
Components	**Q'ty**	**Current (A)**	**Voltage(V)**	**Power (W)**
Raspberry Pi	1	1.1	12	13.2
Camera	1	0.2	12	2.4
Ultrasonic	1	0.02	12	0.24
Camera servo	2	0.3	12	3.6
Sum	5	1.6A	12	19.2W

Table 7. Main Board Power Consumption

Driving Board Power Consumption				
Components	**Q'ty**	**Current (A)**	**Voltage(V)**	**Power (W)**
Drive Board	1	0.1	12	1.2
Motors for Tracks	2	0.8	12	19.2
Sum	3	1.7A	12	20.4W

Table 8. Driving Board Power Consumption

Robot Power Consumption				
Components	**Q'ty**	**Current (A)**	**Voltage(V)**	**Power (W)**
Total	8	3.3A	12	39.4W

Table 9. Robot Power Consumption

Battery Life Analysis				
Components	**Q'ty**	**Capacity (Ah)**	**Current (A)**	**Battery Life (hr)**
Battery	1	6	3.3	1.8

Table 10. Battery Life Analysis

E. *Latency Analysis*

As a search and rescue robot, it is important to ensure reliable, remote operation and a reasonable response time. We measured the request response time and radio strength in a typical scenario. Thus, we were able to find the maximum distance at which the robot can be reliably controlled.

We evaluate this in an open area with the robot and laptop connected to a mobile hotspot provided by a cell phone. For the tests, the robot is placed at different distances (5 meter intervals) from the mobile hotspot. Figure 15 illustrates the response time measurement procedure. We developed a simple python script that sends requests from the laptop to the robot at time t1 and records the server's response time as time t4. If the response isn't received within 2 seconds, the request is marked as timed out. Otherwise, the response time is represented by the equation $\frac{t4-t1}{2}$. One hundred requests were sent and the average response time was calculated as the overall response time at that distance. Similarly, the radio strength at different distances was measured using a Wi-Fi analysis application on another mobile device.

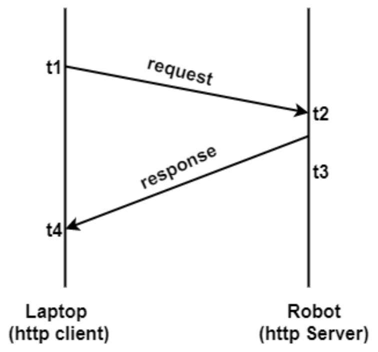

Figure 15. Response Time Measurement

The results of our testing are shown in figure 16. When the robot was at a distance of 70 meters from the mobile hotspot, the radio signal decreased from 0 to -90 mDb, and the packet delay increased from 20ms to 200ms. When the distance increased further, the packet loss increased and had an obvious effect on the operation.

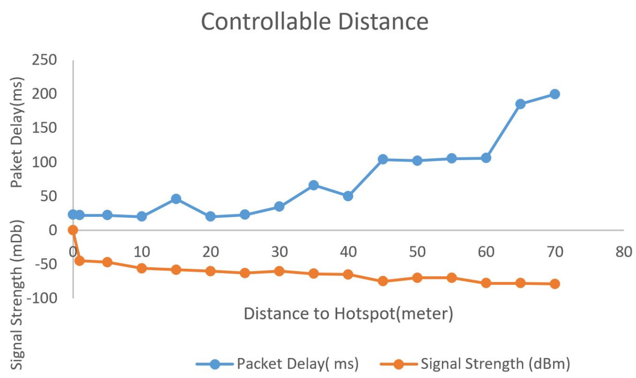

Figure 16. Controllable Distance

V. Planned System

During the Comprehensive Design Review, we focused on the progress made on IntelliSAR's object detection model and demonstrated the model's ability to detect and classify people and rocks with moderate accuracy. Additionally, we discussed our proposed deliverables for the Final Project Review (FPR). These consisted of further improving the accuracy of the object detection, improving the training and evaluation datasets, and improving the robustness of the robot itself. Following the feedback from the Comprehensive Design Review, we adjusted our goals to also include rolling back the inclusion of the rock detection, training the model to detect an additional, less ambiguous object, and improving the semi-autonomous navigation algorithm and related sensors.

The rock detection was implemented for the purpose of demonstrating that our object detection system could classify more than just a person. The camera's rock/obstacle detection was also added with the intention of complementing IntelliSAR's existing semi-autonomous capabilities, as it would help lighten the obstacle avoidance's reliance on the sole ultrasonic sensor. However, we found that the rock detection was not very reliable and seemed to actually decrease the reliability of our person detection as we experimented with this setup. This is likely due to the ambiguity of a rock's shape and how an image of a person's head, with its round shape and lack of colors due to our use of an infrared camera, could be easily mistaken for a rock. For the final product, we would have instead opted to train the model to detect specific human body parts (e.g. human hand) rather than people and rocks.

In regards to the semi-autonomous navigation, we encountered an issue with the ultrasonic sensor that was causing the obstacle avoidance to malfunction occasionally. The ultrasonic sensor that came with the robot parts that we purchased post-CDR was angled upwards and could not be adjusted. This sometimes caused IntelliSAR to fail to detect an obstacle directly in front that was not tall enough to be hit by the ultrasonic sensor's upward-angled waves. To fix this issue, we would have adjusted the default mount to ensure that the ultrasonic sensor was level. Additionally, to further improve the reliability of the obstacle avoidance, it would have likely been ideal to swap out the ultrasonic sensor for a LIDAR sensor to get higher resolution and more accurate distance measurements.

VI. Project Management

We have completed a significant part of IntelliSAR, and our next tasks are to incorporate GPS tracking and improve the object detection and semi-autonomous navigation. When beginning our development of IntelliSAR, the hardware selection and assembly progressed smoothly, with almost all of these tasks being successfully completed in the PDR stage. This progress allowed us to allocate more time towards our subsequent work. Through rapid iterative development, we gradually improved the control functions of the car.

Our team cooperates and works well together. During the first project semester, Yong and Arthur were in China, and Derek was in the United States. Despite this separation, everyone still carried out the effective communication and efficient development needed to ensure the steady progress of our project. Each of us were responsible for our own tasks but would help each other when any problems were encountered. Every week, we held team meetings with our advisor, Professor Tessier, to communicate progress and our next goals. Yong was acting as the team manager and was responsible for the overall hardware development and control functionality of the entire car. Derek was responsible for implementing object detection and semi-autonomous navigation. Arthur was responsible for some web development tasks, sensor settings for IntelliSAR, data collection and analysis, and the MDR poster.

During the second project semester, we continued to maintain the same productive team atmosphere and continued to put our best effort into fulfilling our original goals for IntelliSAR despite the departure of Yong from our team. Derek inherited the role of team manager and was responsible for reconstructing the robot and restoring all prior functionality, compiling the training and evaluation dataset, integrating the Coral USB accelerator, improving the accuracy of the object detection, and re-implementing and improving the semi-autonomous navigation. Arthur was responsible for compiling the training and evaluation dataset, improving robustness of the chassis, collecting data, and performing power analysis.

VII. Conclusion

From the beginning, we have hoped to design a useful product that will improve search and rescue initiatives. Practicality is an important factor that is worth prioritizing, and we have made sure to take this into account in IntelliSAR's specifications. In terms of mobility and maneuverability, IntelliSAR can move at a maximum speed of about 6.5 km/h, climb up slopes of up to 30 degrees, and can cope with most outdoor environments such as mud, grass, and rough roads. In terms of obstacle avoidance, the car can automatically stop and turn when encountering obstacles with a detection distance of about 30 centimeters and a stopping time of about 0.5 seconds. IntelliSAR's camera is able to rotate 180 degrees horizontally and vertically and has night vision capabilities, allowing it to maintain performance in dark environments.

For the Comprehensive Design Review in the second project semester, we reconstructed the robot with all functionality restored, improved the accuracy of the object detection, improved the speed of the object detection, and re-implemented the semi-autonomous navigation and improved its reliability. To complete our project, we planned to complete the following goals: further improve the accuracy of the object detection, improve the training and evaluation datasets, improve the robustness of the robot itself, roll back the inclusion of the rock detection, train the model to detect an additional, less ambiguous object, and improve the semi-autonomous navigation algorithm and related sensors.

Acknowledgment

Special thanks to:

Prof. Russell Tessier for being our advisor and guiding us through technical and non-technical aspects of our project.

Prof. Yadi Eslami and Prof. Arman Pouraghily for being our evaluators and providing helpful, valuable feedback on our project in each review.

Francis Caron for ordering the parts we needed and getting them to us as quickly as possible.

References

[1] T. Holzer and J. Savage, "Global Earthquake Fatalities and Population." Earthquake Spectra, 2013. Available: https://www.earthquakespectra.org/doi/abs/10.1193/1.4000106

[2] T. Drabek, H. Tamminga, T. Kilijanek, and C. Adams, "Managing Multi-organizational Emergency Responses: Emergent Search and Rescue Networks in Natural Disaster and Remote Area Settings." Ctr for the Study and Prevention of Violence Institute for Behavioral Science, 1981. Available: https://www.ncjrs.gov/app/abstractdb/AbstractDBDetails.aspx?id=133298.

[3] R. Murphy, "Marsupial and shape-shifting robots for urban search and rescue," in IEEE Intelligent Systems and their Applications, 2000. Available: https://ieeexplore.ieee.org/document/850822.

[4] R. Murphy, J. Kravitz, S. Stover, and R. Shoureshi, "Mobile robots in mine rescue and recovery," in IEEE Robotics & Automation Magazine, 2009. Available: https://ieeexplore.ieee.org/document/5069840.

[5] J. Walker, "Search and Rescue Robots – Current Applications on Land, Sea, and Air". Emerj, 2019. Available: https://emerj.com/ai-sector-overviews/search-and-rescue-robots-current-applications.

[6] M. Micire, "Evolution and Field Performance of a Rescue Robot." 2007. Available: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.20218.

[7] A. Phunopas, W. Jitviriya, and A. Blattler, "RoboCup Rescue 2018 Team Description Paper iRAP Robot." 2018. Available: https://robocup-rescue.github.io/team_description_papers/2018/Champ2018_Thailand_iRAP_TDP.pdf.

[8] "iRobot 510 PackBot Multi-Mission Robot." Army Technology, 2014. Available: https://www.army-technology.com/projects/irobot-510-packbot-multi-mission-robot.

[9] T. Hornyak, "iRobot military bots to patrol 2014 World Cup in Brazil." CNet, 2013. Available: https://www.cnet.com/news/irobot-military-bots-to-patrol-2014-world-cup-in-brazil.

[10] "Yahboom G1 smart tank robot kit." Yahboom. Available: https://category.yahboom.net/products/g1tank.

[11] "Raspberry Pi 4." Raspberry Pi Foundation. Available: https://www.raspberrypi.org/products/raspberry-pi-4-model-b.

[12] "Yahboom 4WD expansion board for robot car." Yahboom. Available: https://category.yahboom.net/products/4wd-expansion-board.

[13] "Yahboom horizontal ultrasonic sensor distance module avoid obstacle." Yahboom. Available: https://category.yahboom.net/products/horizontal-ultrasonic-sensor.

[14] "DHT11 Humidity & Temperature Sensor." Available: https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf.

[15] "USB Accelerator." Google. Available: https://coral.ai/products/accelerator.

[16] "MakerFocus Raspberry Pi 4 Camera." MakerFocus. Available: https://www.makerfocus.com/collections/rpi-cameras/products/raspberry-pi-4-camera-night-vision-adjustable-focus

[17] "Yahboom 370 Gear motor." Yahboom. Available: https://category.yahboom.net/products/yahboom-370-gear-motor.

[18] "Yahboom SG90 servo." Yahboom. Available: https://category.yahboom.net/products/yahboom-sg90-servo.

[19] K. Sato. "What makes TPUs fine-tuned for deep learning?" Google Cloud. Available: https://cloud.google.com/blog/products/ai-machine-learning/what-makes-tpus-fine-tuned-for-deep-learning.

[20] J. Brownlee, "A Gentle Introduction to Transfer Learning for Deep Learning." 2019. Available: https://machinelearningmastery.com/transfer-learning-for-deep-learning.

[21] "The Confusing Metrics of AP and mAP for Object Detection." mc.ai, 2018. Available: https://mc.ai/the-confusing-metrics-of-ap-and-map-for-object-detection.

[22] J. Huang et al., "Speed/accuracy trade-offs for modern convolutional object detectors." 2017. Available: https://arxiv.org/pdf/1611.10012.pdf.

[23] "Tensorflow Detection Model Zoo." Tensorflow. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md.

[24] "Overfitting in Machine Learning: What It Is and How to Prevent It." Elite Data Science. Available: https://elitedatascience.com/overfitting-in-machine-learning.

[25] "Tensorflow Object Detection API." Tensorflow. Available: https://github.com/tensorflow/models/tree/master/research/object_detection.

[26] "Edge TPU Compiler." Google. Available: https://coral.ai/docs/edgetpu/compiler.

[27] "Open Images Dataset V5." Google LLC. Available: https://storage.googleapis.com/openimages/web/index.html

[28] "OIDv4 ToolKit." EscVM. Available: https://github.com/EscVM/OIDv4_ToolKit.

[29] "LabelImg." tzutalin. Available: https://github.com/tzutalin/labelImg.

[30] "Detection Evaluation." COCO. Available: http://cocodataset.org/#detection-eval.

[31] "Raspberry Pi 4 Tech Specs." Raspberry Pi Foundation. Available: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications.