

PatrolBot

Revised Specification and Design

University of Nevada, Reno
Department of Computer Science and Engineering

Team 08

Jesus Aguilera, Brandon Banuelos, Connor Callister, Max
Orloff, Michael Stepzinski

Instructors: Devrin Lee, Dr. David Feil-Seifer, Vinh Le

Advisors: Dr. Hung La, Dr. Alireza Tavakkoli, Officer Matthew
Stewart

February 18th, 2022

Abstract

The PatrolBot project is about creating a semi-autonomous patrol robot. The PatrolBot project is intended to be an essential tool for campus police forces and other small security operations in properly monitoring their grounds. The project will utilize a robot connected to the internet, a web server, a website, and machine learning models. Our team must implement a web server, control a robot through the internet, build the front-end of our website, connect everything within the backend, and train then deploy the machine learning models. The purpose of this document is to describe our project's specifications and design.

Recent Project Changes

There are no new changes to our project since the last submission.

Updated Specification

Summary of changes in project specification

Since the specification document written last fall, the methods by which we aim to accomplish our project have changed, although the end result will still be the same. This has led to only a few of our requirements changing to reflect our new design. In the document this past fall, all of our requirements were based on having an integrated one-system solution where the server, machine learning models, and user interface would all be on a raspberry pi connected to our robot. Using this plan, we were able to accomplish our goals in last semester's demo, however, our entire system ran much too slow to be usable. To speed up our system, our new version of the project uses a networked approach instead. Our project will still fulfill the same requirements as last semester, new ones reflecting our changed approach are added.

Updated technical requirements

Functional Requirements

Table 1: Functional Requirements for the PatrolBot.

ID	Level	Description
FR00	1	When malicious objects are detected, a medium threat alert will be made on the interface
FR01	1	When malicious objects are near bikes, a high level threat alert will be made
FR02	1	When suspicious activity is detected, a low threat alert will be made
FR03	1	PatrolBot shall identify objects of interest through live video
FR04	1	PatrolBot shall allow users to activate and deactivate the object detection model
FR05	1	PatroBot shall allow user to decide what objects are to be detected from set list of available objects
FR06	1	
FR07	1	The PatrolBot dashboard shall include past recordings accessible from the

		side navigation panel.
FR08	1	PatrolBot shall allow the user to customize the dashboard to their individual liking.
FR09	1	PatrolBot will stream video feed from the robot to the web application
FR10	1	PatrolBot will show estimated robot location
FR11	1	PatrolBot will use odometry readings to help with location tracking
FR12	1	PatrolBot shall operate using a robot and a camera setup
FR13	1	PatrolBot shall have the option to be manually controlled through the web application
FR14	1	Multiple users shall be able to view the PatrolBot dashboard at once
FR15	2	When alerts are made over a certain level, alerts can be sent to users' phones
FR16	2	PatrolBot shall maintain a text log of objects detected and time stamp
FR17	2	The PatrolBot dashboard shall allow account creation.
FR18	2	PatrolBot will utilize a camera that can pan around its environment.
FR19	2	PatrolBot shall use a Kalman filter to fuse GPS and odometry data for location estimation
FR20	2	PatrolBot shall utilize ROS for semi-autonomous navigation
FR21	3	PatrolBot shall traverse campus with full autonomy.
FR22	3	PatrolBot shall deliver web-based notifications through the dashboard when a potential threat is computed.
FR23	3	PatrolBot shall be developed using custom hardware
FR24	3	A separate model will be able to predict the likelihood that an individual will steal
FR25	3	PatrolBot will use multiple robots

Non-Functional Requirements

Table 2: Non-functional Requirements for the PatrolBot.

ID	Level	Description
NFR00	1	PatrolBot will be able to run for at least an hour nonstop
NFR01	1	PatrolBot's dashboard will be easy to navigate
NFR02	1	PatrolBot will be implemented in Python.
NFR03	1	PatrolBot will utilize the tensorflow/pytorch platforms.
NFR04	1	PatrolBot will utilize ROS for robot communication.
NFR05	1	PatrolBot will use a Raspberry Pi 4 as a microcontroller.
NFR06	1	PatrolBot shall have an intuitive and beautiful looking dashboard.
NFR07	1	PatrolBot shall use Ubuntu 18.04 LTS for the robot computer
NFR08	2	PatrolBot will utilize video feed gathered from the robot to determine a potential threat.
NFR09	2	PatrolBot shall use AWS SageMaker to deploy the machine learning model for improved complexity and computational power.
NFR10	2	PatrolBot shall use AWS IoT Roborunner to maintain the robot

Updated use case modeling

Use Case Diagram

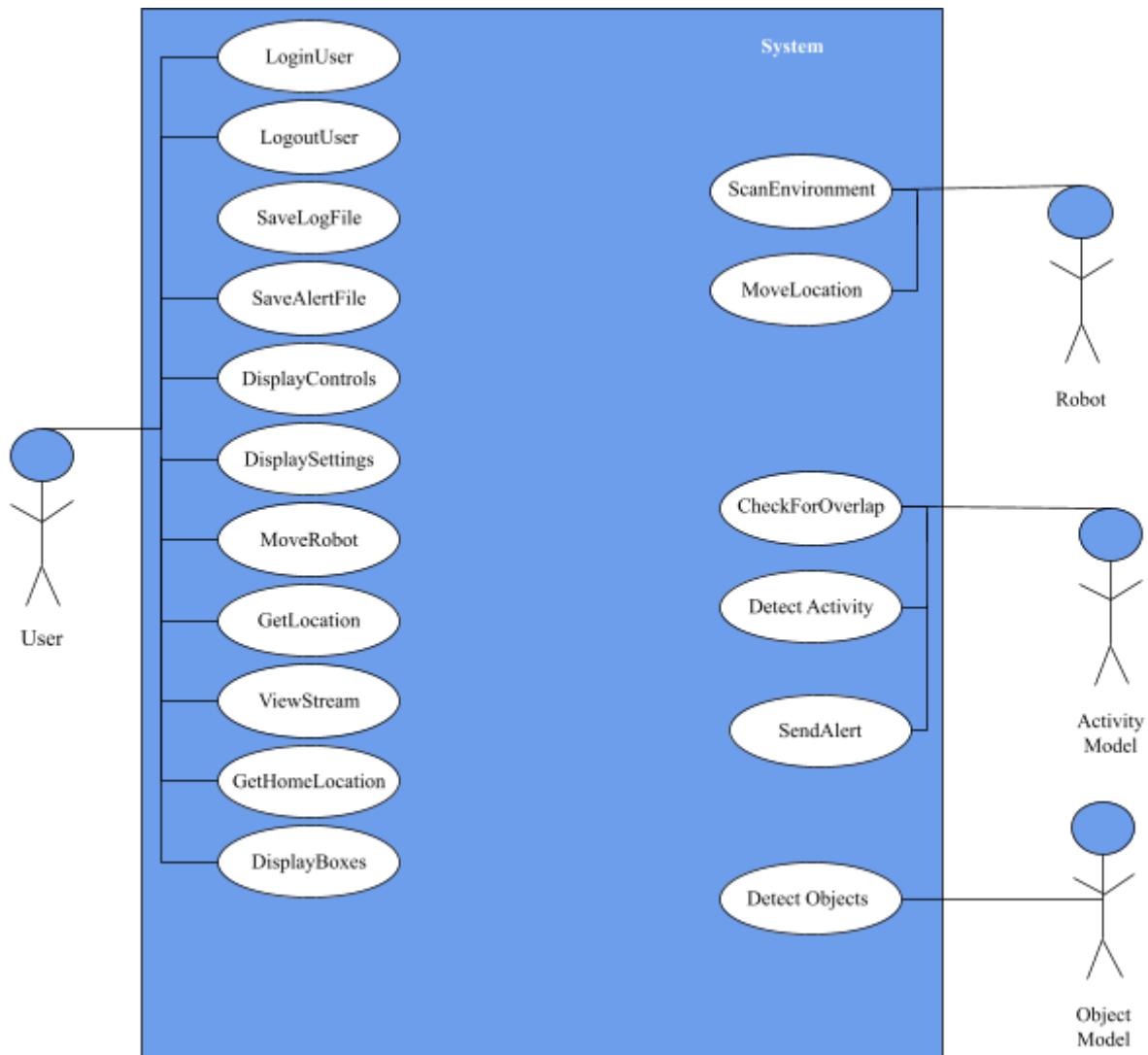


Figure 1: Use case diagram for user, robot, activity model, and object model.

Detailed Use Cases

Table 3: Detailed Use Case Descriptions.

ID	Use Case	Description
UC01	CheckForOverlap	The model takes in video frames as input and checks if a malicious object is close to a bike.

UC02	DetectActivity	The model takes in video frames as input and predicts what activity is occurring.
UC03	SendAlert	If the models and algorithm detect something is wrong, an alert will be presented on the interface.
UC04	LoginUser	Upon accessing the robot dashboard, the user must login to access the controls and data of the robot(s). This login checks for validity of user credentials and either grants or denies access to the robot dashboard.
UC05	LogoutUser	The user may log out from the PatrolBot dashboard and return to the welcome screen.
UC06	SaveLogFile	The user can save the current list of loggable actions from the machine learning model to a separate text file.
UC07	SaveAlertFile	The user can save the current list of computed potential threats detected from the camera feed to a separate text file.
UC08	DisplayControls	The user can switch to the controls page of the program to command the robot.
UC09	DisplaySettings	The user can switch to the settings page of the program to edit specific settings for the robot and program.
UC10	ScanEnvironment	The robot can scan its surroundings using an attached video camera.
UC11	MoveRobot	The user can manually move the robot in a direction.
UC12	GetLocation	The user can get the current estimated robot location.
UC13	ViewStream	The user can view the camera feed of the robot.
UC14	GetHomeLocation	The user can view the location of the robot's charger and base.
UC15	DetectObjects	The model can take in video frames and predict what objects are present.
UC16	DisplayBoxes	The user can decide to enable bounding boxes on the video feed.
UC17	MoveLocation	The robot can turn on the wheels' motors to move itself.

Requirement Traceability Matrix

FR15																	
FR16																	
FR17			■							■							
FR18						■					■						
FR19									■								
FR20			■	■			■	■	■	■					■	■	■
FR21			■	■			■	■	■	■					■	■	■
FR22			■	■			■	■	■		■						
FR23			■	■													
FR24	■	■	■	■						■					■	■	■

Figure 2: This requirements traceability matrix shows the associations between functional requirements and use cases.

Updated Design

Summary of changes in project design

Since last semester, our project design has changed greatly. Instead of a one-system design approach, our project is divided between multiple systems. Our project now consists of 4 separate pieces: the user interface, the backend server, the model processing, and the robot. The user interface will be accessed by means of a website. The backend of that website will run on a web server hosted with AWS. The machine learning models will run on the same server if possible, or a different one if the processing load is too much. The raspberry pi attached to the robot will only have the software necessary for robot controls, camera streaming, and connection to the server. This approach allows everything to run in real-time at an acceptable speed. This approach also makes it easier for team development, since one team member can work on the robot, another can work on the models, another can work on the website, and so on.

Updated high-level and medium-level design

Context Model

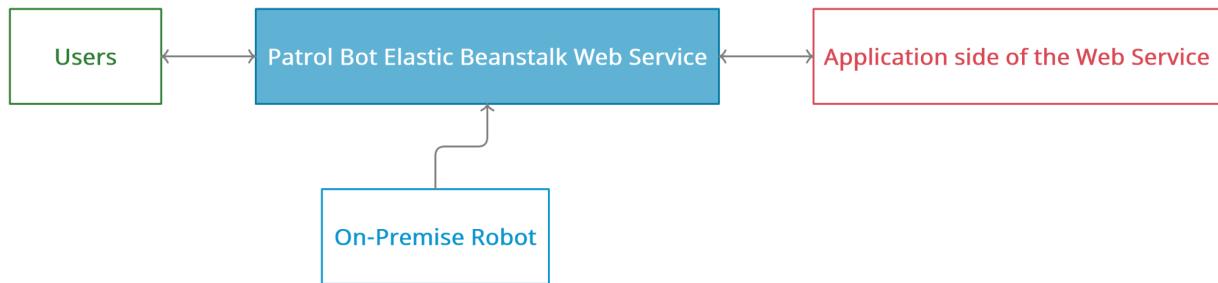


Figure 3: PatrolBot high-level context model

Figure 3 illustrates the high-level context of the PatrolBot system, providing the sub-system relationship of all possible systems. The PatrolBot context model consists of several subsystems ranging from the user dashboard, on-premise robot, and the application side of the web server. The PatrolBot's main system is the AWS elastic beanstalk service. This system is a web application and mimics the central point of the web server by providing a bridge between the users, on-premise robot, and the backend machine learning models. The user dashboard subsystem contains the user viewpoint to all operational consistencies. This includes the login to enter the user dashboard, a main screen viewpoint of the robot's camera feed, a GPS location of the robot, and several other dashboard functionalities. The on-premise robot subsystem consists

of a robot traversing around the university campus and providing a live camera feed of bike racks. The robot will stream the camera feed through its local network and pass it to the web service for the user to see on the PatrolBot dashboard. The application side of the web application will be integrated within the elastic beanstalk web server and will be displayed on the landing page of the user dashboard. This is where the machine learning model will perform its fundamentals on the camera feed.

Program Units

Table 4: The table lists the programming units under the activity model system.

Activity Model System	
save_video()	Input: Camera frames Output: Mp4 file Exceptions: N/A Description: The camera frames get saved into a mp4 file, so the activity model system can be run on the video.
run_model()	Input: Mp4 file Output: Activity and confidence of detection Exceptions: The camera or model crashes Interrupts: A stop command is given Description: The model is run on mp4 files to extract predicted activities recognized.
send_alert()	Input: Threat level Output: Alert on user interface Exceptions: N/A Description: If the model detects threatening actions, it will send a security alert to warn the user.
check_for_overlap()	Input: Bounding box coordinates for bolt cutters or angle grinders and bikes Output: True or False Exceptions: N/A Description: The algorithm checks for IOU between the bounding boxes to determine if the objects are on top of each other.

Table 5: The table lists the programming units under the object detection model system.

Object Detection Model System	
train_model()	<p>Input: Dataset of labeled pictures Output: Trained model Exceptions: N/A Description: There is no trained model for bolt cutters and angle grinders so a custom dataset was created and used to train a model for object detection.</p>
save_video()	<p>Input: Camera frames Output: Mp4 file Exceptions: N/A Description: The camera frames get saved into a mp4 file, so the Object Detection model system can be run on the video.</p>
run_model()	<p>Input: Mp4 file Output: Bounding boxes around detected objects Exceptions: The camera/model crashes, the user turned off bounding box overlay, or the user turned off the model Interrupts: A stop command is given Description: The model is run on mp4 files to extract bounding boxes for detected objects.</p>
add_to_log()	<p>Input: Model output of label of object and time of detection. Output: line of text to log file Exceptions: User turned off specific objects to add to log file Description: The user can determine which detected objects are stored in the log file through settings.</p>
save_logfile()	<p>Input: Objects detected and time stamps Output: .txt file with list of all objects detected and when they were detected on the camera stream Exceptions: N/A Description: The user can decide to save the log file with time stamps to their device.</p>

Table 6: The table lists the programming units under the web server system.

Web Server	
dashboard_view()	<p>Input: WSGIRequest (Web server gateway interface request)</p> <p>Output: Rendered Template</p> <p>Exceptions: If the template page does not exist, throw an error notice or display a 404 page.</p> <p>Description: When a user logs into the user dashboard, returns an http response and fills the page with a dashboard html template page.</p>
welcome_view()	<p>Input: WSGIRequest (Web server gateway interface request)</p> <p>Output: Rendered Template</p> <p>Exceptions: N/A</p> <p>Description: When the user first accesses the page, the welcome page is rendered.</p>
livefe()	<p>Input: WSGIRequest (Web server gateway interface request)</p> <p>Output: Streaming http response</p> <p>Exceptions: Attempts to access camera feed if it exists and generates a frame by frame sequence. If it is not able to be accessed, do not generate an http response.</p> <p>Description: Generates a frame by frame camera feed and displays a live (streaming) http response to be displayed on a webpage.</p>
establish_connection()	<p>Input: IP address (camera or robot gps module)</p> <p>Output: Secure connection</p> <p>Exceptions: Throw error notice when connection is not able to be established.</p> <p>Description: Attempts to establish a secure connection to either the camera for a live feed or robot's gps module to display on the main webpage.</p>

send_movement_command()	<p>Input: Directional movement command</p> <p>Output: Robot receives movement command</p> <p>Exceptions: Robot not activated, robot movement system not initialized, robot did not move</p> <p>Description: This program unit sends a command to the robot to move</p>
-------------------------	--

Table 7: The table lists the programming units under the Robot system.

Robot Systems	
send_power_on_signal()	<p>Input: Physical robot power on button</p> <p>Output: Call to activate_camera() and activate_movement(), True if calls succeeded</p> <p>Exceptions: Battery too low to power on</p> <p>Description: This program unit sends the signal to begin robot functionalities.</p>
activate_camera()	<p>Input: None</p> <p>Output: Streams camera video feed</p> <p>Exceptions: Camera missing</p> <p>Description: This program unit activates the on-board camera</p>
activate_movement()	<p>Input: None</p> <p>Output: Robot movement system is ready for commands</p> <p>Exceptions: Robot not found</p> <p>Description: This program unit activates the robot movement.</p>
send_power_off_signal()	<p>Input: None</p> <p>Output: Calls to deactivate each subsystem, True if calls succeeded</p> <p>Exceptions: Systems not activated</p> <p>Description: This program unit sends the signal for each subsystem to deactivate itself.</p>
recieve_movement_command()	<p>Input: Movement command</p> <p>Output: Robot moves in desired direction</p> <p>Exceptions: Motor error, wheels did not move</p>

	Description: This program unit activates the motors of the robot to get it to move
begin_video_stream()	Input: On command Output: IP address of robot stream Exceptions: Camera not found/accessible Description: This program unit starts the video stream for the camera
end_video_stream()	Input: Off command Output: Video stream ends Exceptions: Cannot end video stream Description: This program unit shuts off the robot video stream

Data Structures

In order to have an account based approach to the PatrolBot dashboard, the team established an accounts database with the help of the Django web framework. The current database setup on the django side is defaulted to using SQLite 3 and is accessible through the web framework using python scripts or SQL queries. The current setup of the database is structured in table 8 with a general example for a record.

Table 8: The main structure to the accounts database of the web server.

id	password	last_login	is_superuser	username	last_name	email	is_staff	is_active	date_joined	first_name
1	[hash]	2022-02-18 01:34:03.947512	1	patrolbot.admin			1	1	2022-02-05 01:18:11.245134	

To train the object detection model, a database has been established. The database is currently made of around 1,000 images total. In this database, there is an image folder and a labels folder. The images folder contains all the images from the four classes the team defined. These classes are people, bikes, bolt cutters, and angle grinders. Each image in this image folder has at least one example of any class to be identified. Accompanying each image in a separate labels folder is around 1,000 text files with the coordinates of bounding boxes for each object to be detected in the corresponding image. An example of these labels is shown in Table 9. Each entry in the text

files has the class of the object in the bounding box, and the coordinates to create the bounding box based on the center, width, and height of the box.

Table 9: An example of a text label for an object detection database image.

Class	X Center	Y Center	Width	Height
1	0.518028846153 8461	0.498798076923 0769	0.90625	0.997596153846 1539

Updated hardware design

Hardware Components Diagram

Figure 4 contains all of the required hardware components of the project and the connections between them. The “center” of the system is a Raspberry Pi 4 which will act as the microcontroller for the Rover Zero, stream the video feed to the web application, and communicate with the GPS module to identify its current location. to process video from the camera connected to the Pi.

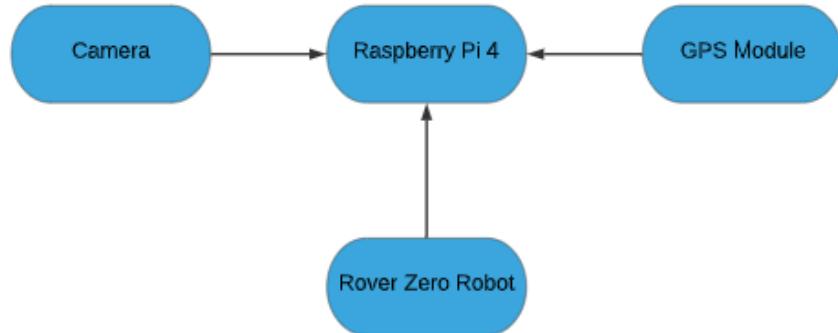
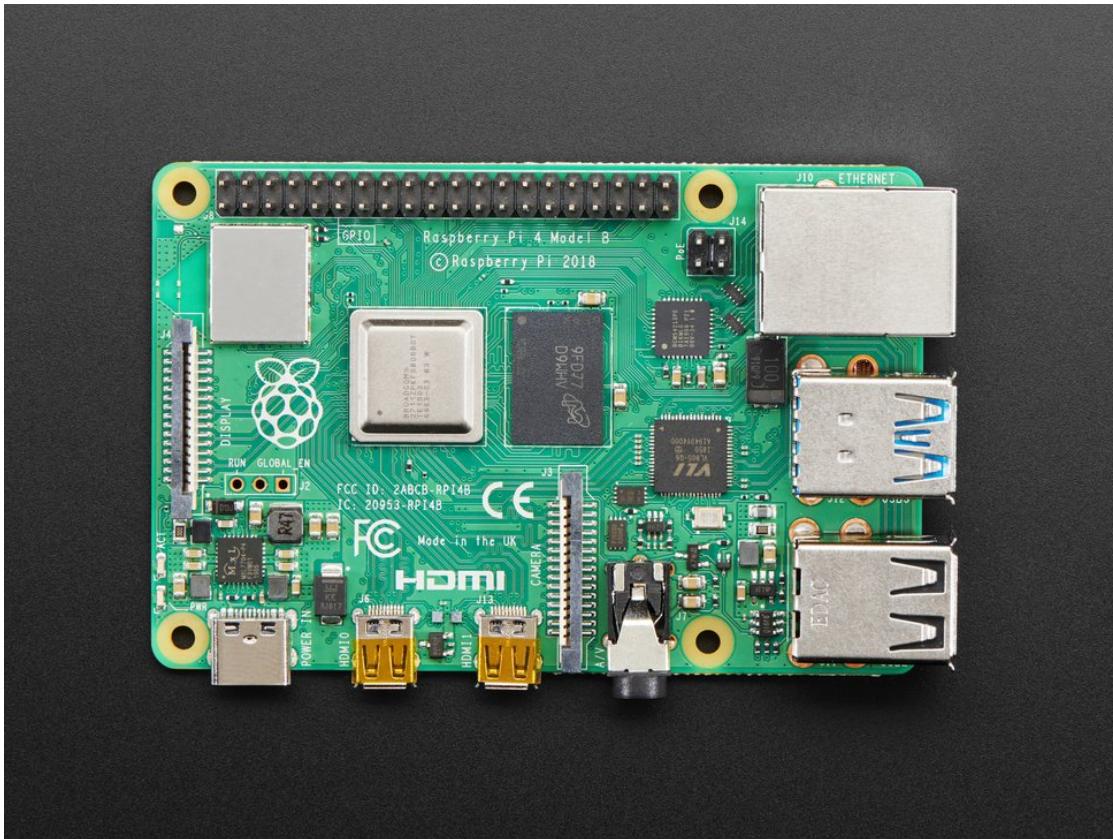


Figure 4: Displays high-level overview of hardware components utilized and their corresponding connections.

Hardware Components Descriptions

Raspberry Pi 4



Snapshot 1: Shows a top-down view of a Raspberry Pi 4.

Image Source:

https://www.adafruit.com/product/4296?gclid=Cj0KCQiA4b2MBhD2ARIAsAIrcB-R_qLp198Y93_7QBNF7LkaKsHP8e1atIkdsy9QEkkmC6JDEiAw2JbEaAl4rEALw_wcB

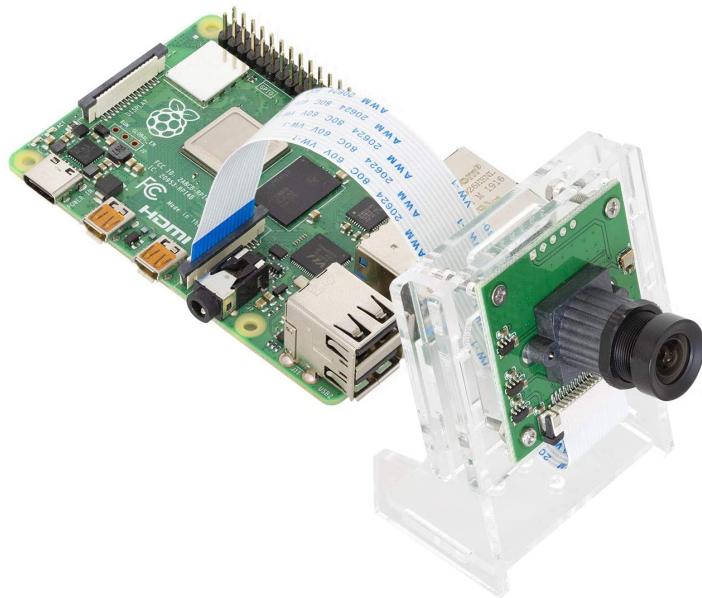
The Raspberry Pi 4 will act as the central system of the project as shown in Snapshot 1. A top-down view of the Pi is shown in Snapshot 1. It will act as the microcontroller for the Rover Zero and provide all necessary functionalities with the camera and GPS modules. It will also act as the “middleman” between the web application and the robot through a shared wireless network connection.

Camera



Snapshot 2: Shows the Arducam for Raspberry Pi Camera Module.

Image Source: https://m.media-amazon.com/images/I/61cnX8IvngL.AC_SL1500.jpg

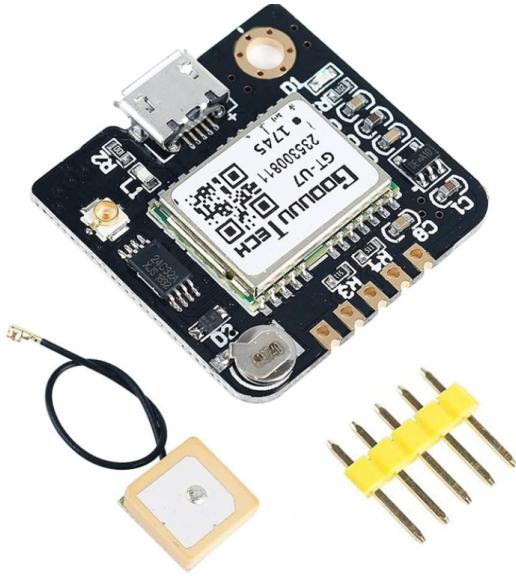


Snapshot 3: Shows how the Arducam is connected to Raspberry Pi 4.

Image Source: https://m.media-amazon.com/images/I/71XozxuTzcL.AC_SL1500.jpg

The camera utilized will be the Arducam for Raspberry Pi Camera Module shown in Snapshot 2. It has a M12 lens to enhance the field of view and allow for higher quality images. It will be connected directly to the Raspberry Pi 4, as shown in Snapshot 3, to allow Pi to stream live video feed it is providing to the web application.

GPS Module



Snapshot 4: Shows the GPS Module GPS NEO-6M

Image Source: https://m.media-amazon.com/images/I/71D+goSD3CL.AC_SL1500.jpg

The GPS Module will connect directly to the Raspberry Pi 4 to allow the Pi to understand its location which will result in understanding the robot's location. This GPS data will be filtered with the odometry data of the robot to provide a more accurate estimation of the robot's location.

Rover Zero



Snapshot 5: Shows the fully assembled Rover Zero robot.

Image Source:

https://cdn.shopify.com/s/files/1/0055/0433/5925/products/Zero4WD_Pictures_0006_Zero4WD_RobotOnly.png?v=1644961498

The Rover Zero is a fully assembled terrestrial robot equipped with a differential drive motor system. It includes a built in lithium ion battery, motor driver, motors, and encoder. It is also equipped with built-in software using ROS to allow for basic communication for a user. Its battery will power the Pi via USB connection and it will communicate with the Pi's commands using ROS and send odometry data to the Pi for localization estimates. It will house all of the above hardware components on its chassis.

Updated user interface design

Figure 5 shows the initial landing page of the web server main system. This is a separate URL from the project description website and is used to access the user dashboard.

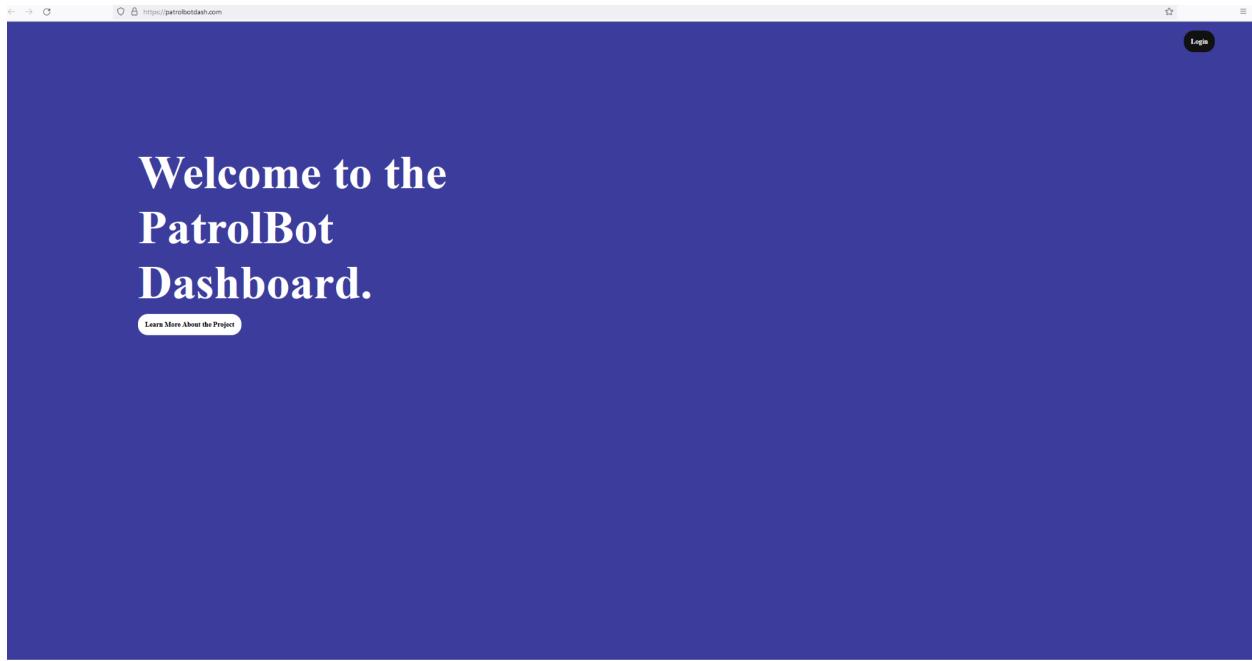


Figure 5: PatrolBot welcome screen.

Figure 6 shows a button that will redirect the user to the detailed project website and is displayed in the landing page when first attempting to access the user dashboard. This is for when the user wants to learn more about the main core of the project before accessing the dashboard.



Figure 6: Button that will route the user to the project website.

Figure 7 shows a button that will login the user into the user dashboard and is displayed in the welcome screen.

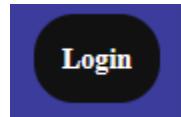


Figure 7: Login button.

Users will then be greeted with the login form after clicking the login button. From here, the user will be able to access the user dashboard after entering the correct superuser account information. At the time, since there is only one robot there will only be one user in the database. This is so no other users can create an account without having access to a robot.

A screenshot of a web browser window showing the "Admin Login" page. The URL in the address bar is "https://patrolbotdash.com/accounts/login/". The page has a dark blue header and body. The main content area is a white box titled "Admin Login". It contains two input fields: "Username:" with the value "patrolbot.admin" and "Password:" with a redacted password. Both fields have small "1" icons next to them. Below the fields is a large dark blue "LOG IN" button.

Figure 8: Login page for the dashboard.

Figure 9 shows the landing page after the user successfully logs into the user dashboard.

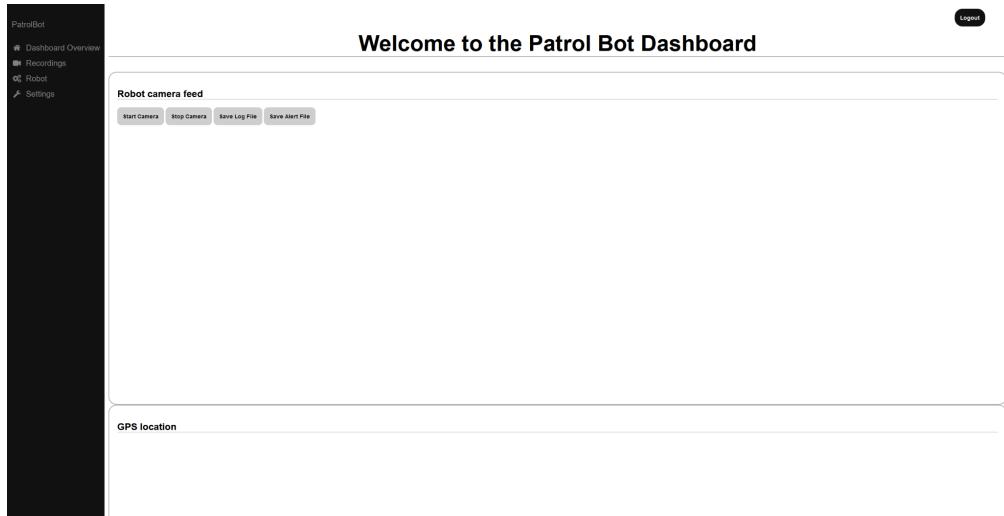


Figure 9: Dashboard landing page after a successful login.

The logout button that will log the user out of the user dashboard. This button is displayed on every page of the dashboard so the user is able to log out from anywhere.

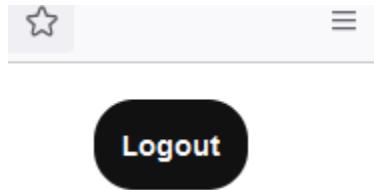


Figure 10: Logout button that appears on every screen on the dashboard.

The side navigation panel that will redirect the user to several other accessible pages. As the project progresses and new ideas become available, this side navigation panel will grow with more options.

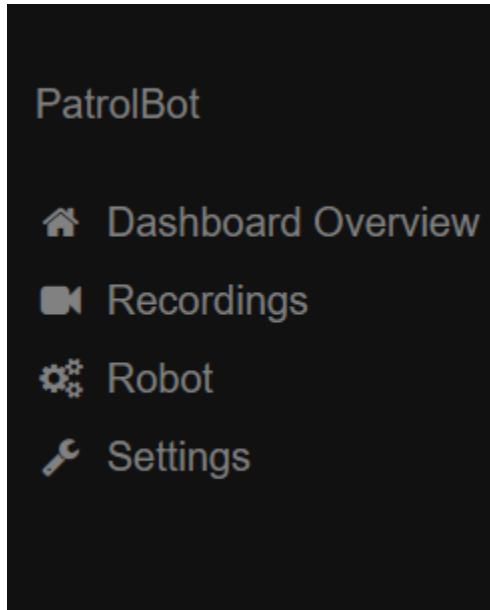


Figure 11: Side navigation panel for the dashboard.

The robot camera feed controls displayed on the landing screen of the user dashboard.

Robot camera feed



Figure 12: Camera feed controls for the camera mounted on the on-field robot.

When the user clicks the robot option in the side navigation panel, the user will be redirected to a page displayed in figure 13.

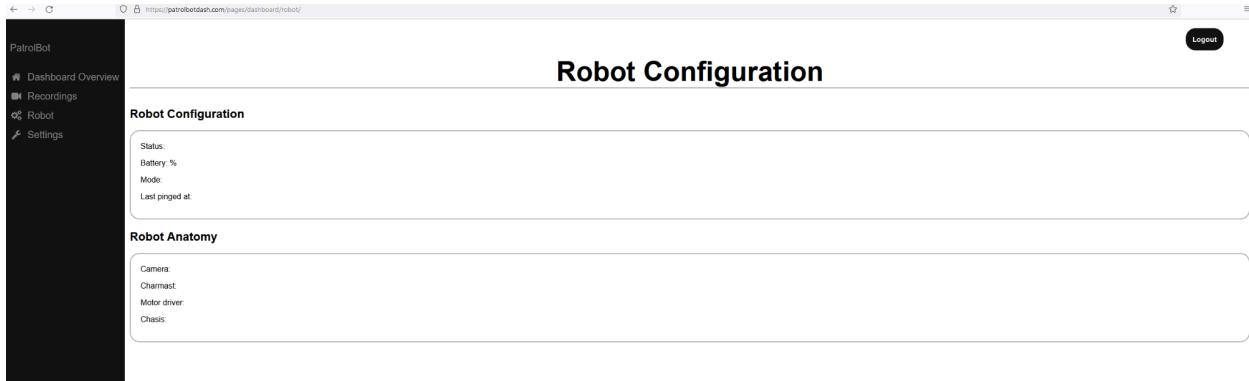


Figure 13: Robot configuration landing page accessed by the side navigation panel.

The current robot configuration settings that will display helpful information about the robot. At the time, it will only display the status (online or offline), the battery percentage, the traversal model (auto or manual) and the last pinged location in latitude and longitude coordinates.

Robot Configuration

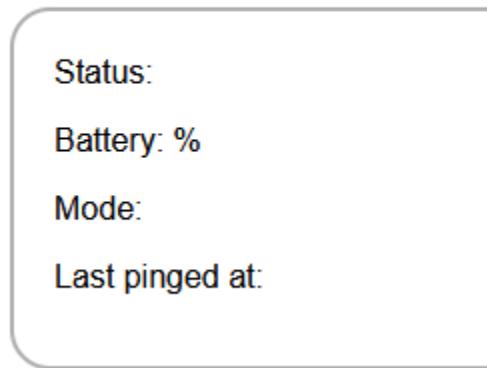


Figure 14: Robot configuration panel that will display information about the current state of the robot.

The robot anatomy information that will display each part of the robot.

Robot Anatomy

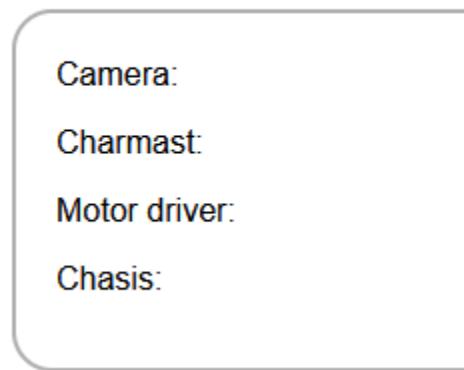


Figure 15: Robot anatomy panel that shows each part of the robot.

The dashboard settings accessed from the side navigation panel.

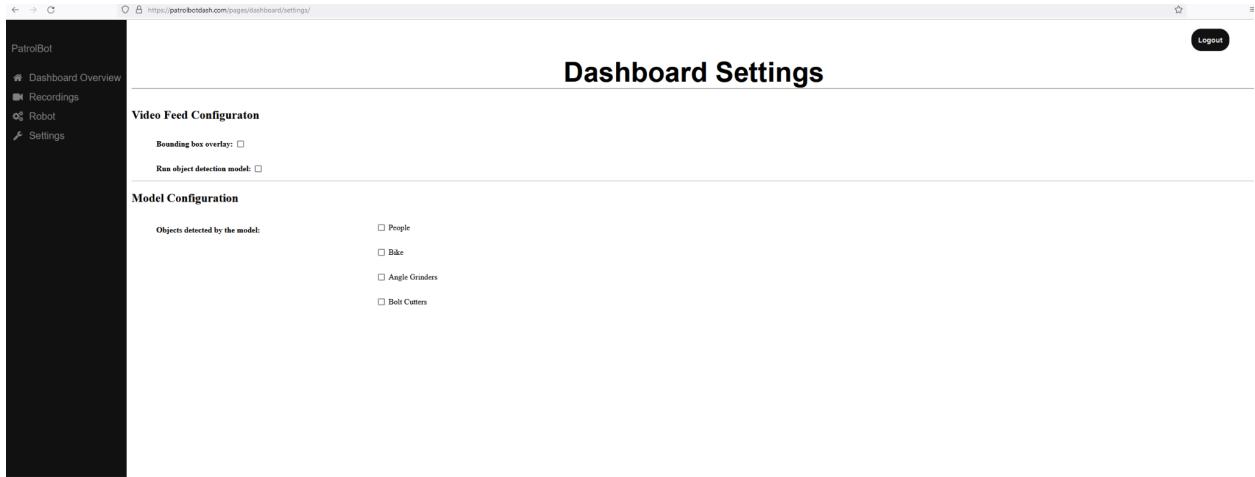


Figure 16: Dashboard settings page accessed by the side navigation panel.

Figure 17 shows the video configuration settings for the main screen of the user dashboard. At the moment, it will allow the user to disable or enable the bounding box object detection overlay on the camera feed and disable or enable the objection detection model itself.

Video Feed Configuration

Bounding box overlay:

Run object detection model:

Figure 17: Video feed configuration option panel.

Figure 18 shows the configurable settings to the model itself. This allows the user to determine whether the model should classify certain objects for the machine learning model.

Model Configuration

Objects detected by the model:

People

Bike

Angle Grinders

Bolt Cutters

Figure 18: Model configuration option panel.

Glossary of Terms

1. **Backend** - The part of a web application that users do not see.
2. **Frontend** - The part of a web application that users see.
3. **Robot Operating System (ROS)** - A popular set of open-source software libraries and tools used to develop robotic applications.
4. **Model** - Machine learning algorithms trained on data, used to refer to a specific set of trained algorithms.
5. **Deep Learning** - Using a neural network with hidden layers to train models on your data.
6. **Machine/Computer Vision** - Use of camera, computer hardware, and software algorithms to automate visual inspection tasks.
7. **Angle of View (AOV)** - Visual inspection depending on the size of the robot.
8. **Autonomous Mobile Robot (AMR)** - Robot that utilizes advanced sensors, computer vision, and machine learning to navigate its environment.
9. **Degrees of Freedom** - Robots ability to move in a single independent direction of motion.
10. **Neural Network** - Machine learning algorithm to mimic the way the biological neurons signal.
11. **Machine Learning (ML)** - the use and development of computer systems that are able to learn and adapt without following explicit instructions, by using algorithms and statistical models to analyze and draw inferences from patterns in data.
12. **Training** - In machine learning, training is a process in which an ML algorithm is fed with sufficient data to learn from.
13. **Validation** - In machine learning, model validation is referred to as the process where a trained model is evaluated with a testing data set. The testing data set is a separate portion of the same data set from which the training set is derived.
14. **Testing** - In machine learning, model testing is referred to as the process where the performance of a fully trained model is evaluated on a testing set.
15. **Feedback** - The return of information from a manipulator or sensor to the processor of the robot to provide self-correcting control of the manipulator.
16. **YOLO (You Only Look Once)** - An algorithm that provides real time object detection
17. **Angle grinder** - A handheld electronic tool with a circular blade meant for grinding or cutting
18. **Bolt cutters** - A long-handled tool with blades meant for cutting chains and padlocks
19. **Alert** - An event the robot(s) deem necessary to notify the user about
20. **Threat** - An event that the machine learning models running on the robot deems suspicious
21. **Inflated 3D Network** - A neural network that relies on training based on multiple frames at a time rather than single images.

22. **Temporal Segment Network** - A neural network that breaks down a video into segments with convolutional networks running on each to combine towards an aggregated prediction.
23. **Kinetics400** - A dataset of at least 400 video clips for 400 different actions
24. **GPS Module** - A hardware component that sends GPS location info to it's connected device.
25. **Raspberry Pi** - A small computer that plugs into various devices to allow for affordable computing projects.
26. **Odometry** - Data acquired from motion sensors to estimate a change in position over time.
27. **Differential Drive** - A two-wheeled driving system with independent actuators for each wheel that allows for the wheels to spin at different speeds.
28. **Localization** - The process of determining where a robot is located with respect to its surrounding environment.
29. **Intersection Over Union (IOU)** - A ratio of area of intersection of two boxes over the area of union.

Engineering Standards and/or Technologies

Object Detection

PatrolBot is using YoloV5 to perform object detection. YoloV5 is a neural network that can be trained to detect multiple classes of objects.

Activity Detection

PatrolBot is using MxNet and GluonCV to perform action recognition. These libraries provide implementations of temporal segment networks and inflated 3D networks trained on activity datasets. These models will allow PatrolBot to record videos and extract a list of possible activities being performed to help with threat detection.

Amazon Web Services / AWS

AWS is a subsidiary of Amazon providing on-demand cloud computing platforms and APIs to individuals, companies, and governments, on a metered pay-as-you-go basis. PatrolBot will utilize AWS to do all of the computing for the object detection and activity detection models.

PYQT5 / PYQT Designer

PyQt is a Python binding of the cross-platform GUI toolkit Qt, implemented as a Python plug-in. PatrolBot is utilizing PYQT5 and their drag and drop GUI designer, PYQT Designer, to build our UI.

Django

Django is a high-level Python web framework that enables rapid development of secure and maintainable websites. Django is being used to build PatrolBots web application.

Universal Serial Bus / USB

It is an established **standard** for specifying requirements for cables, connectors, and various protocols relating to these connections.

In the project a USB cord will be utilized to provide a connection between our robot and a Raspberry Pi 4 to allow the Pi to act as a microcontroller for the robot.

IEEE 802.11ac / WiFi

This is the current **standard** for wide area networks that utilize wireless network communications.

The project will utilize WiFi to establish communication between our robot and our web application through a Raspberry Pi 4.

Robot Operating System / ROS

This is a **technology** that is an open-source set of software libraries used for the purpose of creating robot related applications.

The project will utilize ROS to control the actions, specifically movement, of the robot.

Project impact and context considerations

Social Impact

The social impact of the Patrol Bot would include a safer community for bikes on the UNR campus as well as a safer community in general. Having a bike stolen from you on campus could be quite devastating if this is your only form of transportation. The Patrol Bot would reduce these situations for students and faculty.

Safety Impact

PatrolBot would help make UNR's campus safer. Theft often involves a confrontation if the victim sees the thief committing the crime. Because bike thieves would be deterred from committing crimes on campus, it would be less likely that physical altercations arise on campus.

Environmental Impact

PatrolBot would allow the UNR police department to survey areas without the need of vehicles. The past reliance on a deployed officer roaming campus in their police vehicle results in constant emissions from their vehicle during their shift. A switch to the use of a robot would lessen these emissions.

Economic Impact

PatrolBot would allow for a greater coverage of space without the need of employing more officers and utilizing fewer existing officers. After the initial cost of purchasing a fleet of PatrolBots, future funding could be spent policing different crimes, improving existing technology, or professional development for officers.

Updated List of References

Problem Domain Book:

Casasent, D. P., Hall, E. L., & Röning, J. (2003). Intelligent robots and computer vision XXI : algorithms, techniques, and active vision : 28-29 October, 2003, Providence, Rhode Island, USA.

This book details various applicable algorithms for robotic machine vision. This includes efficient models for mobile intelligent robots built using the fundamentals of deep learning and computer vision. This book also highlights techniques that will reduce computational complexity in performing these various algorithms on robot-captured videos using ‘active vision’ as well as finding the optimal camera calibration parameters.

Reference Articles:

Kevin Carey, Benjamin Abruzzo, Christopher Lowrance, Eric Sturzinger, Ross Arnold, and Christopher Korpela "Comparison of skeleton models and classification accuracy for posture-based threat assessment using deep-learning", Proc. SPIE 11413, Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications II, 1141321 (21 April 2020); <https://doi.org/10.1117/12.2556422>

This is a paper that compares different pose detection models used for a threat detection algorithm. Patrol Bot will most likely use one of these pose detection models for our own algorithm and this paper gives good information about differences between them.

Kushwaha, A., Kolekar, M., & Khare, A. (2012). Vision based method for object classification and multiple human activity recognition in video surveillance system. Proceedings of the CUBE International Information Technology Conference, 47–52.
<https://doi.org/10.1145/2381716.2381727>.

An algorithm for video surveillance systems that helps better analyze human behavior. Using object classification and a human activity recognition model, this algorithm can capture specific human activities from a live and dynamic surveillance system. This conference proceeding can be used as a reference and standard framework for our projected robot that intends to analyze human behavior using the same approach.

Fang, M., Li, L., Xu, H., Zhang, F. (2018). Movement human actions recognition based on machine learning. *International Journal Of Online Engineering*, 14. [10.3991/ijoe.v14i04.8513](https://doi.org/10.3991/ijoe.v14i04.8513)

This journal discusses using region convolutional neural networks to classify images of human activity into different categories such as walking, running, and jumping. Additionally, it discusses using these classifications for security purposes to detect items such as theft or fighting. Both of these apply directly to the PatrolBot and provide a solid knowledge base for the project moving forward.

N Bhuvaneswary, S Pravallika, V Jayapriya, & K Himabindu. (2021). Night Surveillance Military Spy Robot using Raspberry Pi. *Annals of the Romanian Society for Cell Biology*, 25(5), 5740–5747.

This journal article describes a night surveillance military spy robot that performs various tasks to elevate security for soldiers. Built using a raspberry pi, this spy robot detects mines, human activity, and is developed using a night surveillance web camera. This journal article highlights the importance of developing the robot using a camera specifically for night-time surveillance and how such a system can be built with a card-sized computer.

Project Related Websites:

<http://wiki.ros.org/>

This website is the primary resource for simulating and implementing robotics functionalities for the project. It contains tutorials to learn ROS that will help the team members assigned to the robotics section of the project develop the tools needed to reach our goals. It also will act as a resource throughout implementation, expansion, and testing the project's robotic aspects.

<https://www.tensorflow.org/>

This website will be the primary resource for implementing our deep learning algorithms to create a usable model in Python. It includes tutorials to help the machine learning team members get familiar with its functionalities. It also has a vast number of resources and thorough documentation for all of its associated libraries.

<https://cv.gluon.ai/contents.html>

This website provides tutorials for the implementations of activity recognition models as well as other models which can perform object detection or pose estimation. This will help with the development of the threat model.

<https://roverrobotics.com/>

This website provides all the necessary documentation for the robot used in the project. It will help the team understand the specifications of the robot, built-in software and hardware, and act as a guide for implementing all robot functionalities.

Contribution of Team Members

Michael

Tasks: Cover page, abstract, recent project changes, summary of changes in project specification, summary of changes in project design, 3 level 1 FR, 1 level 2 FR, 1 level 3 FR, 1 NFR, 4 use cases, 4 program units

Time: 4 hours

Max

Tasks: 3 level 1 functional requirements, 1 level 2 functional requirement, 1 level 3 function requirement, 3 nonfunctional requirements, 4 use cases, 4 programming units, Social Impact, 3 Technologies, 2 Reference Articles, Traceability Matrix, editing and formatting

Time: 5 hours

Connor

Tasks: 3 level 1 functional requirements, 1 level 2 functional requirement, 1 level 3 functional requirement, 2 nonfunctional requirements, 4 use cases, 4 programming units, 5 glossary terms, 2 standards, 1 technology, environmental and economic impacts, 1 journal, 1 resource website, 1 problem domain book, high level hardware overview diagram and hardware component descriptions

Time: 5 hours

Brandon

Tasks: 3 level 1 functional requirements, 1 level 2 functional requirement, 1 level 3 function requirement, 3 nonfunctional requirements, 4 use cases, Use Case Diagram, 4 programming units, Safety Impact, Object Detection Database, 1 Project Related Website, 2 Technologies

Time: 4 hours

Jesus

Tasks: 3 level 1 functional requirements, 1 level 2 functional requirement, 1 level 3 functional requirement, 3 nonfunctional requirement, 4 use cases, context model image and description, 4 programming units, website database description, User Interface design.

Time: 4 hours