# AE 5098 - Directed Research

Michael Beskid
michael.beskid@gmail.com

Spring 2024

## Overview

This directed research project was aimed at developing a platform for conducting real-world experiments with a simulated threat field. A projector network was set up to display a composite image on a large area of the floor space in the lab. This setup allows for the creation of a varying spatio-temporal field, where the colors projected are representative of gas concentration. A mobile robot is equipped with a camera system that is capable of measuring and interpreting the simulated threat field. A simple demonstration was completed to showcase the capabilities of this platform.

This document contains detailed information about the various subsystems mentioned in the introduction. A quick start guide is provided at the end of the document which gives instructions for starting up the system and running an experiment. All of the code and other files for this project can be found on Github: https://github.com/WPI-AVMI/threat-field-simulator.

## Equipment/Login Information

### Wi-Fi Network

The NETGEAR R6080 WiFi router in the lab was configured to provide a network for this research project. The dual-band router provides both 2.4G and 5G connectivity. The login information for the network is provided below.

Table 1: Wi-Fi Network Information

| **SSID** | **Password** |
|---|---|
| AVMI-LAB | letmeinplease |
| AVMI-LAB-5G | letmeinplease |

The network settings can be configured by opening the page at IP address 192.168.1.1.

Static IP addresses were assigned for the various devices on the network for convenience. A list of these IP address reservations is given below. These devices can be easily accessed from the rugged laptop via SSH.

Table 2: Static IP Address Assignments

| **Device Name** | **IP Address** |
|---|---|
| AVMI-LAB-RUGGED | 192.168.1.10 |
| AVMI-LAB-01 | 192.168.1.11 |
| AVMI-LAB-02 | 192.168.1.12 |
| AVMI-LAB-03 | 192.168.1.13 |
| AVMI-LAB-04 | 192.168.1.14 |
| AVMI-LAB-CAM-SYS | 192.168.1.15 |
| AVMI-TB4-01-RPI | 192.168.1.20 |
| AVMI-TB4-01-CREATE3 | 192.168.1.21 |

### Rugged Laptop

The control computer for the experimental setup is a Dell Latitude 7330 Rugged Extreme Laptop. The laptop has been configured with a dual boot of Windows 11 and Ubuntu 22.04. The login information for this computer is the same for both operating systems.

Table 3: Rugged Laptop Login Information

| Device Name | Username | Password |
|---|---|---|
| AVMI-LAB | AVMI-LAB-USER | letmein |

Everything for this research project was done within the Ubuntu environment. Ubuntu 22.04 is a Linux distribution which is compatible with ROS2. ROS is a middleware that is used for wireless communication with mobile robots over the network. ROS2 humble is the distribution used in this project.

### Raspberry Pis

Table 4: Raspberry Pi Login Information

| Device Name | Username | Password |
|---|---|---|
| avmi-lab-01 | pi | letmein |
| avmi-lab-02 | pi | letmein |
| avmi-lab-03 | pi | letmein |
| avmi-lab-04 | pi | letmein |
| avmi-cam-sys | avmi-cam-sys | letmein |

## Projector System

### Hardware Setup

The projector system is composed of four projectors which work together to display one large composite image on the floor of the lab workspace. The projectors on mounted on tripods which are arranged around the perimeter of the workspace. The projectors are numbered 1 through 4 as shown in the projector numbering diagram. The current projector model (Optoma ML750ST) has a throw ratio of 0.8 and an aspect ratio of 16:10. The location, height, and pitch angle of the tripods can be adjusted as needed to align the four projectors properly to display the combined image. Zoom, focus, and color settings for each projector can be accessed from the onboard menu to adjust the image quality.

Each projector is controlled by a raspberry pi which is connected via HDMI. The raspberry pis are mounted to the tripods with 3D printed clamp mounts, which also support the inverters for the projector power supplies and extension cords. The CAD files for these mounts are included in the GitHub repository. The raspberry pis are intended for headless operation and are controlled remotely from the rugged laptop. The pis must be powered via the USB-C port.

The system works by generating a color image of the threat field on the rugged laptop, splitting that image, sending the split images to the raspberry pis, and then finally displaying these on the projectors. The image is first generated and split using the Python scripts detailed below. The four split images corresponding to the four quadrants of the original image are saved to a shared network folder. This folder was manually mounted on all four raspberry pis using NFS such that all devices on the network can access the contents of this folder. On the rugged laptop, this folder can be found at /var/nfs/projector-pics. The shared folder is mapped to the directory /nfs/projector-pics for each of the raspberry pis. The raspberry pis each run a Python script which searches for the correct image in this folder and then displays it in full screen on the projector with the FEH image viewer application. The pis will continually check the folder and refresh the display when a new image is uploaded to allow for dynamically changing fields.

### Threat Field Image Generation

The projected images are created with the Threat_Field_Generator.py script which can be found in the Documents/Scripts directory on the rugged laptop. This script imports many helper functions from the file GCS_Functions.py. The script works by first computing a normalized value within the range 0 to 1 for every pixel within in image of a defined size. These values are then mapped to colors to produce a PNG image representing a visualization of the underlying data.

For testing purposes, the script was designed to generate a bivariate normal distribution (2D Gaussian) with several adjustable parameters. First, the value of the specified function is computed for the (X,Y) location of every pixel in the image to generate the data. These values are then used to assign the hue values of corresponding colors using the HSV representation. The HSV values are converted to RGB values to set the color of each pixel in the image. Collectively, this generates a color-coded visualization of the value of the function across all points in 2D space. Several parameters may be configured to change the resulting distribution, such as the location of them mean, variance along both axes, or rotation angle. In the future, it would be possible to produce visualizations of real gas concentration data by providing a matrix of values as input in place of the Gaussian function.

After generating the visualization of the thread field, some additional processing is required to display the field on the projectors. The full image is divided into four quadrants and each component image is saved with a different filename and placed in the shared NFS drive. The images for the top quadrants (projectors 1 and 2) are flipped 180 degrees because they will be projected from the opposite side of the workspace as the bottom quadrants (projectors 3 and 4).

## Camera System

### Components

The camera system is compromised of a raspberry pi 4 and four USB webcams. The system is powered by a portable power bank which powers the raspberry pi via the USB-C port. The two portable power banks in the lab are interchangeable so one can charge while the other is in use.

The webcams are connected to the USB-A ports on the raspberry pi and should be recognized automatically. To check whether the webcams are properly connected to the raspberry pi, SSH into the pi and use the command "v4l2-ctl –list-devices" to show the connected USB devices. The four webcams should be shown at the bottom of the list. Each device will be listed as "UVC Camera" and should list a path of the form "/dev/videoX."

The webcams are secured to the Turtlebot4 with 3D printed mounts which attach to the screw holes on the robot chassis intended for the standoffs for the optional mounting plate. The raspberry pi for the camera system is also mounted on this structure. The camera mounts were designed such that the four cameras are arranged in a radially symmetric pattern about the robot, each an equal distance from the center point and directed at 45 degree angles. The cameras are numbered 1 through 4 in a counterclockwise fashion as indicated in the camera numbering diagram.

### Camera System ROS Node

The raspberry pi for the camera system has a ROS workspace in the "ros2_ws" folder. runs a ROS node called "cam_sys" which is contained in the ROS package "avmi_lab_cam_sys." This node offers a ROS service called "get_color_vals" which is defined by the custom GetColorVals service message type. When this service is called from another node (i.e. the robot), it returns values from the four webcams.

The get_color_vals service is active after the node is launched. This service utilizes the OpenCV library to get the color values from the cameras. First, the script connects to one camera and captures a still image. The script then computes the average RGB values for a square of pixel values located at the center of the image. The RGB values are then converted to HSV, from which the hue value is extracted and normalized between 0 and 1. This is effectively an inverse of the encoding process used to generate the image of the threat field. The process is repeated for all four cameras, and the values are returned to the node which made the service call.

Starting the cam-sys node remotely from the rugged laptop can be accomplished by opening a new terminal and running the following commands:

$ sudo su
$ cd ros2_ws
$ source /opt/ros/humble/setup.bash
$ source install/setup.bash
$ ros2 run avmi_lab_cam_sys cam_sys

# Mobile Robot

## Turtlebot4 Platform

This project makes use of the Turtlebot4 mobile robot platform. The Turtlebot4 is a two-wheel, differential-drive robot. The robot has two onboard computers which are a Raspberry Pi 4 and a Create3. These devices must both be connected to the Wi-Fi network to operate the robot. The Turtlebot4 uses ROS2 Humble to communicate with the rugged laptop. The Turtlebot4 automatically launches the bring-up node automatically upon power up, so the user does not need to run any nodes on the robot manually. The robot communicates with remote devices over ROS topics to report information from sensors and to receive actuation commands. The Turtlebot4 may be tele-operated with the game controller joysticks as explained in the turtlebot documentation.

## ROS Packages

The robot is controlled by running ROS nodes on the rugged laptop. The ROS workspace on the laptop is contained in the folder "turtlebot4_ws". This workspace currently contains two ROS packages. "avmi_lab_interfaces" contains custom message and service definitions that are used within other packages in the workspace. The "avmi_lab_threat_field" package contains the nodes used for controlling the robot for various experiments. The simple demonstration of the simulator capabilities makes use of the "threat_field_navigator" node which is contained within this package. ROS nodes can be run on the robot by opening a terminal and executing the command $ ros2 run ¡package_name¿ ¡node_name¿. An example of this is shown in the quick start guide at the end of this document.

The threat_field_navigator node provides basic navigational abilities and lots of important underlying functionality. This node sets up publishers and subscribers to interact with the robot. Specifically, the node subscribes to the "/odom" topic to get information about the robot's current state, and publishes to the "cmd_vel" topic to send movement commands. Some basic movement functions have been written to command desired velocities and to facilitate more complex maneuvers like driving to a specified point in the workspace. This node also sets up a service call to the get_color_vals service provided by the camera system, and provides a handler to process the camera data upon receipt. For any future work, it is recommended to start from a copy of this node to maintain these critical functions as described.

The goal of the threat_field_navigator node is to direct the robot to the maximum of the threat field. This is accomplished by using data from the camera system to essentially measure the local gradient at the robot's current position in the field. From the four camera readings, the program selects the two highest readings and performs linear interpolation to compute a new heading direction in the direction that the value of the field is increasing. The robot then performs a point turn to orient itself in this direction and drives forward for a short distance. This process is repeated such that the robot will make a service call to the camera system to sense the field, perform some computation, then execute a movement to get closer to the goal. This functionality is shown in the simple demonstration videos in the GitHub repository.

There is significant room for improvement in the performance of the mobile robot. More sophisticated motion control techniques and navigation algorithms could be implemented to improve driving performance. Termination conditions could also be introduced to stop the robot after some goal is accomplished. Many variations could be made to this starter code to achieve the desired functionality needed for conducting various experiments with this real-world simulation platform.

## Running an Experiment

This section provides a quick start guide to running an experiment with this simulation platform. Conducting an experiment can be done in three simple steps as described below.

1. **Start the projector system.** Power on all four projectors and raspberry pis. From the rugged laptop, open a terminal and change to the Scripts directory ($ cd Documents/Scripts). Then run the shell script to start up the display with the command $ ./init_projectors.sh. If any errors occur where the images are not found, power cycling the raspberry pis should fix the issue. The projector system can be shut down by running the script $ stop_projectors.sh.

2. **Start the camera system.** Connect a power bank to the raspberry pi for the camera system. Allow it to connect to the lab WiFi network and then SSH into the device from the rugged laptop. Follow the commands given in the camera system subsection to run the "cam_sys" ROS node. The camera system can be stopped by interrupting the terminal with "ctrl" + "C".

   **Note:** The script "init_cam_sys.sh" was intended to automate this process but fails due to the inability to access root privileges on the raspberry pi when accessing it remotely thought a shell script. If resolved, steps 1 and 2 can be combined by simply running the "init_threat_field.sh" script to start up the projectors and the camera system simultaneously. Similarly, the "stop_threat_field.sh" script can be executed to stop both the projectors and camera system.

3. **Run a program on the robot.** Place the Turtlebot4 on the floor of the workspace. Open a new terminal on the rugged laptop and run the ROS node for whichever program you would like to execute. For example, the simple demonstration program which navigates the robot to the maximum value of the threat field can be run with $ ros2 run avmi_lab_threat_field threat_field_navigator. The robot program can be stopped by interrupting the terminal with "ctrl" + "C".