DEVELOPMENT OF A LOW-COST UNMANNED GROUND VEHICLE FOR CAMPUS SECURITY

Liang Ruiqi[1], Tan Jia Hui, Joy[1], Tay Jing Xuan[2]

[1]Hwa Chong Institution, 661 Bukit Timah Road, Singapore 269734
[2]DSO National Laboratories, 12 Science Park Dr, Singapore 118225

Abstract

This project focuses on developing a low-cost Unmanned Ground Vehicle (UGV) to enhance campus security, addressing the challenges posed by the large landscape of schools during after-hours surveillance. Unlike expensive commercial UGVs, our solution costs only 2.5% of the typical price while retaining essential functionalities.

The UGV is equipped with an encoder motor system, an inertial measurement unit, a camera, a 2D LiDAR, an Arduino Uno Rev 3 and a reComputer J3011 with NVIDIA Jetson Orin Nano 8GB. It integrates Simultaneous Localization and Mapping (SLAM) for navigation and obstacle avoidance, a web-based Ground Control System (GCS) for remote monitoring, and an object detection model based on YOLOv8 for identifying humans and weapons. The GCS provides real-time control and live camera feeds, allowing security teams to monitor and manage the UGV remotely.

Our research demonstrated effective performance in detecting objects, following paths, and navigating the environment, with an improved detection model optimized for low-light conditions. While the current iteration meets basic security requirements, future developments can explore advanced autonomy, path-planning algorithms, and enhanced decision-making through multi-sensor fusion. This project showcases a scalable and affordable solution for improving campus security using robotics and machine learning. A video demonstration of our UGV can be viewed here: https://youtu.be/hpgCD0SYuLE.

1. INTRODUCTION

School security guards are tasked for surveillance duties during after school hours. However, the large landscape in our school makes it infeasible for security guards to roam every corner. As such, unmanned surveillance vehicles are widely used for enhancement of safety.

The benefits of using Unmanned Ground Vehicles (UGV) are profound. As compared to Closed Circuit Television (CCTV) systems that only captures live footage, a UGV can be trained to proactively recognise and follow trespassers. However, according to an article by IFSEC Insider (accessed December 2024) UGVs are often unaffordable for schools, with commercial ones reaching up to USD$70,000.

2. AIMS AND OBJECTIVES

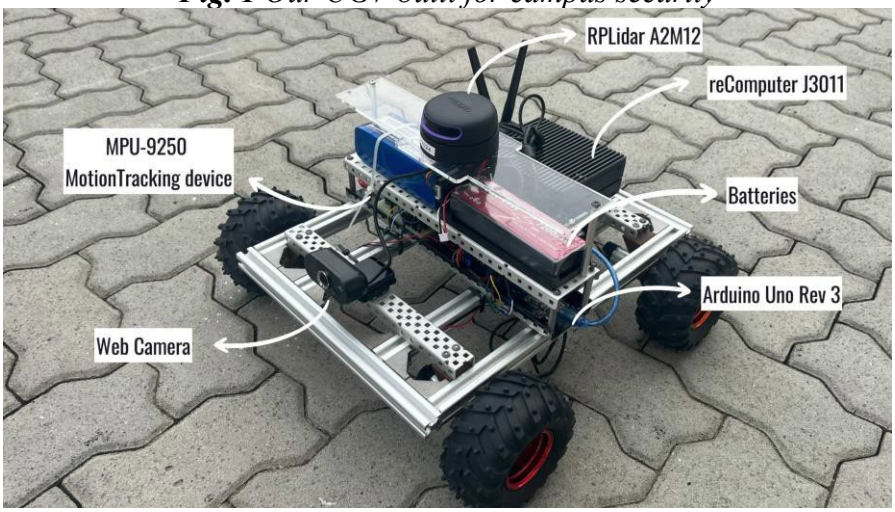
Through this research, we hope to be able to develop a low-cost UGV that will help to patrol the school at night, where only security guards should be present. We will be developing our UGV for the following functionalities:

1. Teleop control of basic movements of the UGV by user through a Ground Control System (GCS);
2. Accurately follow set path chosen by the user;
3. Send live footage back to the user and detect any abnormalities such as human movement and/or dangerous weapons.

3. MATERIALS AND METHODS

3. 1 Hardware

Fig. 1 *Our UGV built for campus security*



The UGV chassis consists of a frame made of 2020 T-slot aluminium extrusions. It is fitted with a set of 34:1 metal gearmotor with encoder, connected to a pair of L298N motor drivers and an Arduino Uno Rev 3 as its microcontroller. An MPU-9250 Inertial Measurement Unit (IMU) is used for orientation data. These are supplemented with an USB Web Camera, a RPLIDAR A2M12 360 degree 2D LiDAR and a reComputer J3011 with Jetson Orin Nano 8GB. Images of individual components of the UGV are shown in Appendix A.

Encoder motors are adapted to provide information about an electric motor shaft's speed and/ or position (according to Encoder Products Company, accessed December 2024). This data is

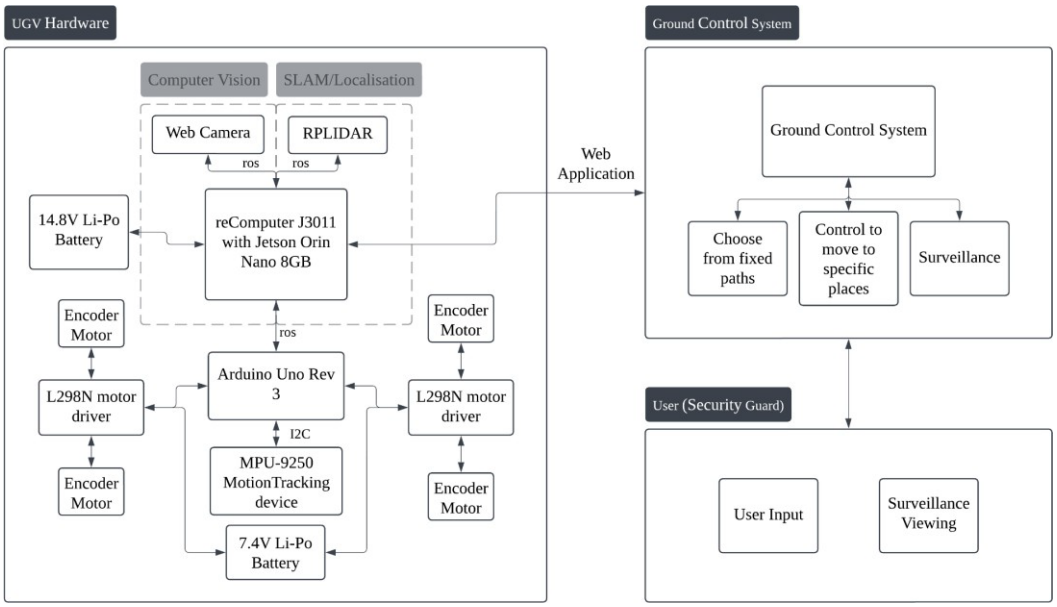
used to accurately control distances to travel and help to track how much the UGV has moved. Next, the MPU-9250 is capable of measuring acceleration, angular velocity, magnetic inclination and temperature (according to TDK InvenSense, accessed December 2024), allowing the UGV to make turns accurately and precisely.

For the camera, a simple USB Web Camera was used as our Machine Learning model and the reComputer's computer vision capabilities are sufficient for object detection. Next, we used the RPLIDAR A2 M12, a 2D LiDAR with a range of 16 metres. Capable of painting a picture of its current surroundings, it can help to localise the UGV within a specified map and prevent it from hitting obstacles in its path.

For our microcontroller, the Arduino Uno Rev 3 was chosen as it is low-cost and has the capacity to support this UGV's needs. It allows us to control motor power, read encoder output and communicate with the MPU-9250 IMU. The reComputer is the onboard computer and is capable of performing video analytics and object detection (according to Kiwi Electronics, accessed December 2024). It receives camera data and acts as the centre of communication between the different components using Robot Operating System (ROS).

3. 2 Architecture Diagram

Fig. 2 *Architecture Diagram*



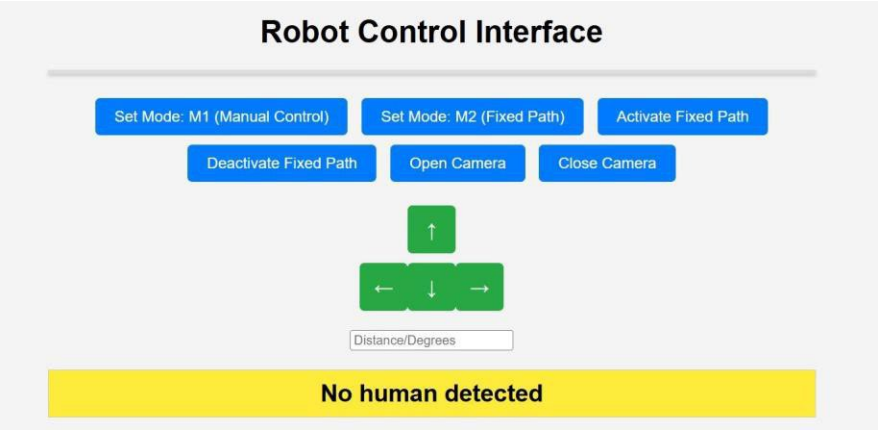
This is the architecture diagram of our UGV. It shows briefly how the different components connect with each other, and how the UGV connects to the Ground Control System operated by the security team.

4. RESULTS AND DISCUSSIONS

The system developed for this UGV is integrated with ROS *(Figure 2)*. The paragraphs below will describe the different capabilities of our system and how they come together as a whole.

4. 1 Ground Control System

Fig. 3 *Web Application*

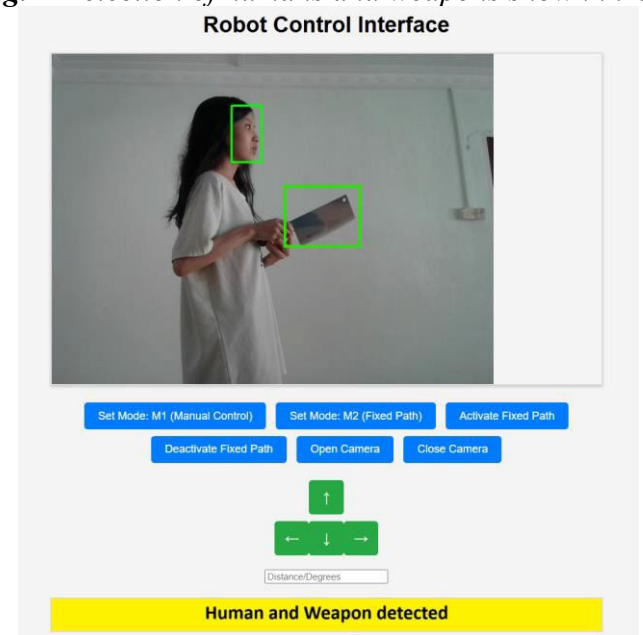


Our Ground Control System (GCS) is a web application that allows the user to view the camera's output and select between the different modes the UGV can take. The GCS allows the security team to remotely control the movement of the UGV and view what the camera sees. The GCS is built using Python and Flask, and it runs on the reComputer. When users click on the buttons on the GCS, a request will be sent to the Flask server. The Flask server processes the input and takes the corresponding action, such as controlling the camera or sending movement commands to the Arduino. Upon receiving the commands, the UGV will execute the desired action.

4. 2 Camera & Object Detection

When the button "Open Camera" is clicked, the OpenCV library grabs frames from the camera which are converted into bytes to be displayed on the web application. As for object detection, each frame is converted into a blob and passed through a computer vision model trained to detect humans and weapons.

Fig. 4 *Detection of humans and weapons shown in GCS*



When humans and weapons are detected by the model, bounding boxes will be drawn around the object and a warning will be displayed onto the screen, notifying the security team. The development of our model is further discussed in the section Machine Learning.

4. 3. 1 Mode 1: Manual Control

By clicking on the green arrows on the web application (front, back, left, right) and entering in the distance to move or degrees to turn (Figure 3), a corresponding command will be sent to the Arduino. Users are then able to send specific commands to control the UGV's movement.

4. 3. 2 Mode 2: Fixed Path

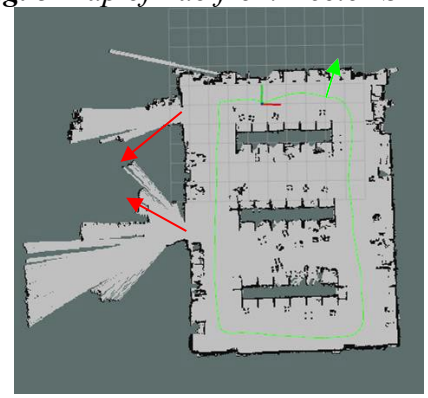
Mode 2 allows the UGV to follow a fixed path pre-determined by the security guard to allow it to patrol and its charging location. An Arduino program was written to take in commands and control the UGV to move a specified distance and direction (such as Front 2m, Left 90^{o}), allowing it to follow a fixed path on loop. However, the UGV often went off course due to drift.

To reduce the effect from drift, SLAM (discussed in the next section) was used to determine the UGV's current location. An algorithm was written [Appendix B] to control the UGV to follow a set of waypoints along the patrol path. The algorithm computes the UGV's desired heading and distance to the next waypoint and sends the corresponding movement command. The command is recomputed whenever an updated pose of the UGV is received, and once the UGV has reached the current desired waypoint, the desired waypoint will be set to the next waypoint in the path. This ensures that the UGV is able to correct its drift and follow the desired path accurately.

4. 4 Simultaneous Localisation and Mapping (SLAM)

SLAM is a method used for autonomous vehicles that allows a map to be built and for the vehicle to be localised in the map at the same time (according to MathWorks, accessed December 2024). Due to the large size of our campus, SLAM is useful in telling security guards where the UGV is currently located. Moreover, it can also help the UGV detect if there are any obstacles ahead.

Fig. 5 *Map of Lab from Hector SLAM*



We chose to use Hector SLAM for our SLAM method as it only requires LiDAR scans as input and can efficiently give the security team a map of the UGV's surroundings, its current pose and the path it has moved. For our experiment, our UGV made a round in our school's lab and it was able to compute its current location accurately and generate a map, as shown in Figure 5.

Hector SLAM helps security guards to visualise the surroundings of the UGV, allowing them to have greater knowledge of the surroundings when taking control over the UGV. It also gives the security team a better idea of where the UGV is within the large campus. SLAM was also

used to ensure that the UGV reaches set poses on the map when following a specified path, mitigating the effects of drifting.

4. 5 Machine Learning

A computer vision model was trained to detect humans and weapons. Using Roboflow, 6000 images of weapons (knives, guns) as well as images of humans in varying backgrounds (urban and nature) and varying distances (near and far) were sourced online and annotated. This process involved drawing bounding boxes around target objects and generating text files that contained coordinates of the bounding boxes. A YOLOv8s model was then run with the images and labels for 30 epochs. As shown below, the model performed well with precision, recall and mean average precision (mAP) scores reaching values of about 0.8 to 0.9.

Fig. 6 *Model Performance for humans*

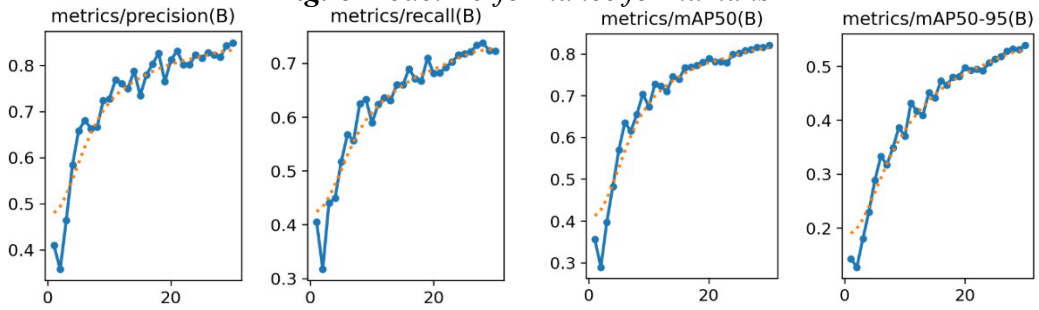


Fig. 7 *Model Performance for weapons*

```
Model summary (fused): 168 layers, 3006623 parameters, 0 gradients, 8.1 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:15<00:00,  3.12s/it]
                   all        154        162      0.922      0.885      0.926      0.617
                  fire        154         50          1      0.897      0.978      0.525
                  guns        154         33      0.943      0.909      0.974      0.681
                hammer        154         25      0.919       0.92      0.934      0.709
                knives        154         31      0.839      0.838      0.831      0.551
                   rod        154         23      0.908      0.859       0.91      0.616
```

As our UGV would be used in low light settings, we decided to improve the model by training it on images with a lower brightness level. This was done using the Python Imaging Library (PIL) which allowed for mass image processing. All images used to train the model were set to a brightness level of 0.3 to stimulate dark environments. The model was then retrained on these darker images. The new model performed better with higher accuracy, as shown below.

Fig. 8 *Results of New Model*

Model / Metrics	Precision	Recall	mAP 50	mAP 50-95
Original Model	0.859	0.719	0.803	0.531
Improved Model	0.863	0.722	0.821	0.539

5. CONCLUSION & PROPOSED WORK

In conclusion, we have successfully developed a cost-effective unmanned ground vehicle (UGV) that retains key functionalities of commercial models while costing only 2.5% of their price *[Appendix C]*. Designed specifically to augment our campus security, the UGV is equipped to patrol the school at night, detect the presence of humans and weapons, and relay real-time information to the security team via an integrated web application. This allows for

seamless remote monitoring and control, significantly enhancing situational awareness and operational efficiency.

Looking ahead, our UGV demonstrates potential for further development. Future iterations could focus on advanced autonomy, such as enabling the UGV to independently plan and navigate routes to target locations while dynamically detecting and avoiding obstacles. Implementing path-planning algorithms like Dijkstra's (referencing W3Schools, accessed December 2024), A*[1] or D*[2] could greatly enhance its efficiency in complex environments. Furthermore, incorporating features like multi-sensor fusion and advanced machine-learning models could improve the UGV's decision-making capabilities, making it an indispensable tool for campus security and other applications.

Attached here, is a link to a YouTube video showcasing the development of our UGV: https://youtu.be/hpgCD0SYuLE

BIBLIOGRAPHY OF REFERENCES

[1] Abiy, T., Pang, H., Tiliksew, B., Moore, K. Khim, J. (n.d.) A* Search.

[2] Stentz, A. 1994. The D* Algorithm for Real-Time Planning of Optimal Traverses.

Appendix A: UGV Components

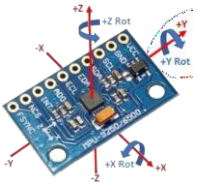
Pololu 25D Metal Gearmotors with encoders	
L298N motor driver	
MPU-9250 MotionTracking device	
Web Camera	
RPLIDAR A2M12 2D laser scanner	
Arduino Uno Rev 3	
ReComputer J3011 with Jetson Orin Nano 8GB	

Table. 1 *UGV Components*

Appendix B: Code for calculating UGV's movement to set poses

```
def compute_command(self):
        if self.current_target_index >= len(self.targets):
            rospy.loginfo("All targets reached.")
            return

        # Get the current target coordinates
        target_x, target_y = self.targets[self.current_target_index]

        while True:
            # Calculate the difference in position
            dx = target_x - self.current_x
            dy = target_y - self.current_y

            # Calculate the required yaw to face the target
            target_yaw = atan2(dx, dy)
            yaw_error = target_yaw - self.current_yaw

            # Normalize yaw_error to the range [-pi, pi]
            yaw_error = (yaw_error + pi) % (2 * pi) - pi

            # Convert yaw_error to degrees
            yaw_error_deg = degrees(yaw_error)

            # Calculate the distance to the target
            distance = sqrt(dx**2 + dy**2)

            # Publish turn command if yaw error is significant
            if abs(yaw_error_deg) > 3.0:
                turn_command = f"R {round(yaw_error_deg)}"
                self.send_to_arduino(turn_command)
                self.executing_command = True
                rospy.sleep(5)
                self.executing_command = False
                break

            # Publish forward command if distance is significant
            elif distance > 0.1:
                forward_command = f"F {round(distance)}"
                self.send_to_arduino(forward_command)
                self.executing_command = True
                rospy.sleep(5)
                self.executing_command = False
                break

            # If the robot is close enough to the target, break the loop and move to the next target
            else:
                rospy.loginfo(f"Target {self.current_target_index + 1} reached: ({target_x}, {target_y})")
                self.current_target_index += 1
                break
```

Appendix C: Cost of UGV

Item	Cost (SGD)
reComputer J3011 - Edge AI Computer with NVIDIA® Jetson™ Orin™ Nano 8GB	$849.10
Slamtec RPLIDAR A2M12 360 Degree 2D Lidar Sensor Kit	$341.93
Arduino Uno Rev 3	$35.77
2 x Motor Driver Dual H Bridge L298N	$13.60
4 x 34:1 Metal Gearmotor 25DX67L mm HP 6V with 48 CPR encoder	$308.00
MPU9250 Motion Tracking Device	$22.60
1080 2K HD Webcam	$50.00
Elements 7200mAh 1000C 7.4V Lipo Battery (Hardcase)	$70.00
Turnigy 5000mAh 30C 18.4V Lipo Pack	$43.99
Chassis Materials	$20.00
Total Cost	**$1754.99**

Table. 1 *Breakdown of UGV component cost*