CG2111A Engineering Principle and Practice
Semester 2 2024/2025

"Alex to the Rescue"
Design Report
Team: B02-6B

Name	Student #	Sub-Team	Role
Karthikeyan Vetrivel	A0307459W	Software	Colour Sensor / Robot Movement
Joshua Yeo Wee Tze	A0309329Y	Firmware	Claw Design / LIDAR implementation
Rajeshprabu Sidharth	A0302676A	Hardware	Circuit Design / Robot Movement
Goh Shao Ann	A0307779L	Software	LIDAR Implementation

Section 1 System Functionalities

Our robot "Alex" is designed to simulate the core functionalities of a search and rescue robot. Teleoperated via a wireless connection, the operator can move Alex through the simulated lunar environment of "Moonbase CEG".

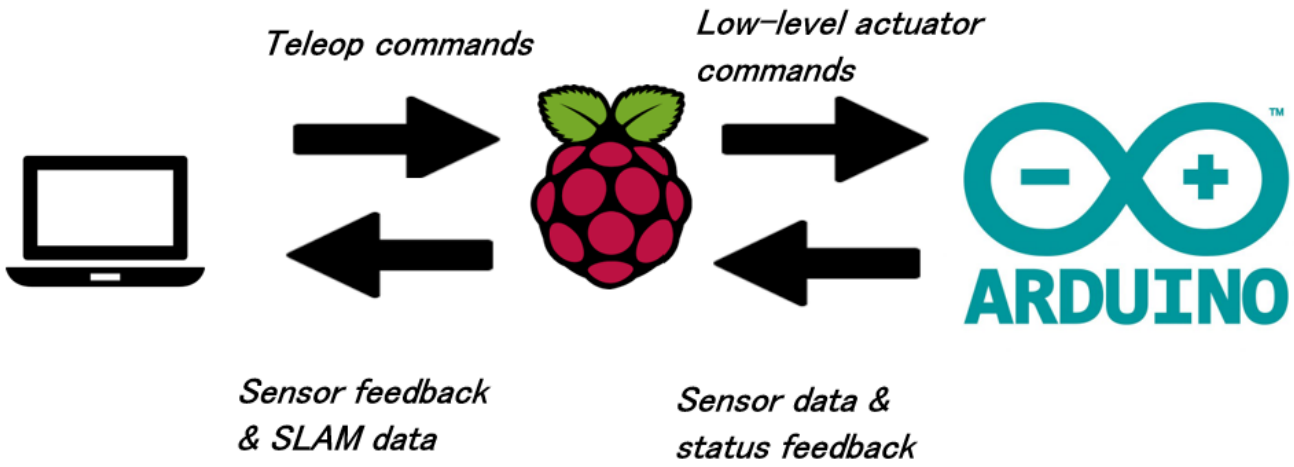
Alex consists of a Raspberry Pi and an Arduino Mega. The Raspberry Pi is the "brains" behind the system, it runs a Master Control Program (MCP) that receives teleoperation commands from the operator using a laptop. It then translates these into low-level actuator commands, which are communicated to the Arduino Mega using the UART port. The Arduino acts as the motor controller. It also processes data received from the ultrasonic sensors and sends it to the Raspberry Pi, thus enabling environmental navigation. At the same time, the Raspberry Pi processes data received from the LIDAR sensor to build a map of the environment. The user interacts with the system using a laptop, connecting to the Raspberry Pi using SSH for motor, claw, and sensor control purposes, and using ROS networking to visualize and obtain SLAM data.

Simulated Lunar Environment
Our Alex robot would be placed in a room filled with various obstacles such as walls, boxes, and tables. As operators, we would teleoperate Alex to locate two astronauts, one marked red and one marked green. Upon finding the red astronaut (injured and trapped), Alex would use its actuator claw to perform a simulated rescue. For the green astronaut (healthy but trapped), Alex would deliver a medpak, completing its objective in the simulated lunar environment.

Core Functionalities:

- Move orthogonally (Forwards, Backwards, Left Turn, Right Turn)
- Identify objects within the test area using the Colour Sensor
- Map out the test area with the help of the LIDAR and SLAM
- Have enough battery life to last the entire test duration
- Be able to be remote-controlled by a teleoperator
- Use an actuator claw to interact with the environment
- Drag the red astronaut (injured) to a safe zone using the claw
- Deliver a medpak to the green astronaut

Diagram:



Section 2 Review of State of the Art

Among the many modern designs for Search and Rescue (SAR) robots, these two stood out to us due to their unique capabilities and design.

RAPOSA:

RAPOSA is a semi-autonomous robot developed for SAR missions primarily in hazardous outdoor environments, such as collapsed structures and debris fields. It consists of a variety of sensors to help accomplish this mission, namely three conventional cameras, a thermal camera, explosive and toxic gas sensors, temperature and humidity sensors, inclinometers, artificial lighting, a microphone, and speakers. RAPOSA is primarily controlled through teleoperation, using a wireless link between the robot and a remote console operated via a conventional GUI and gamepad. The robot runs its control system on an onboard computer and supports both wireless and tethered communication (Silva, Almeida, & Lima, 2006).

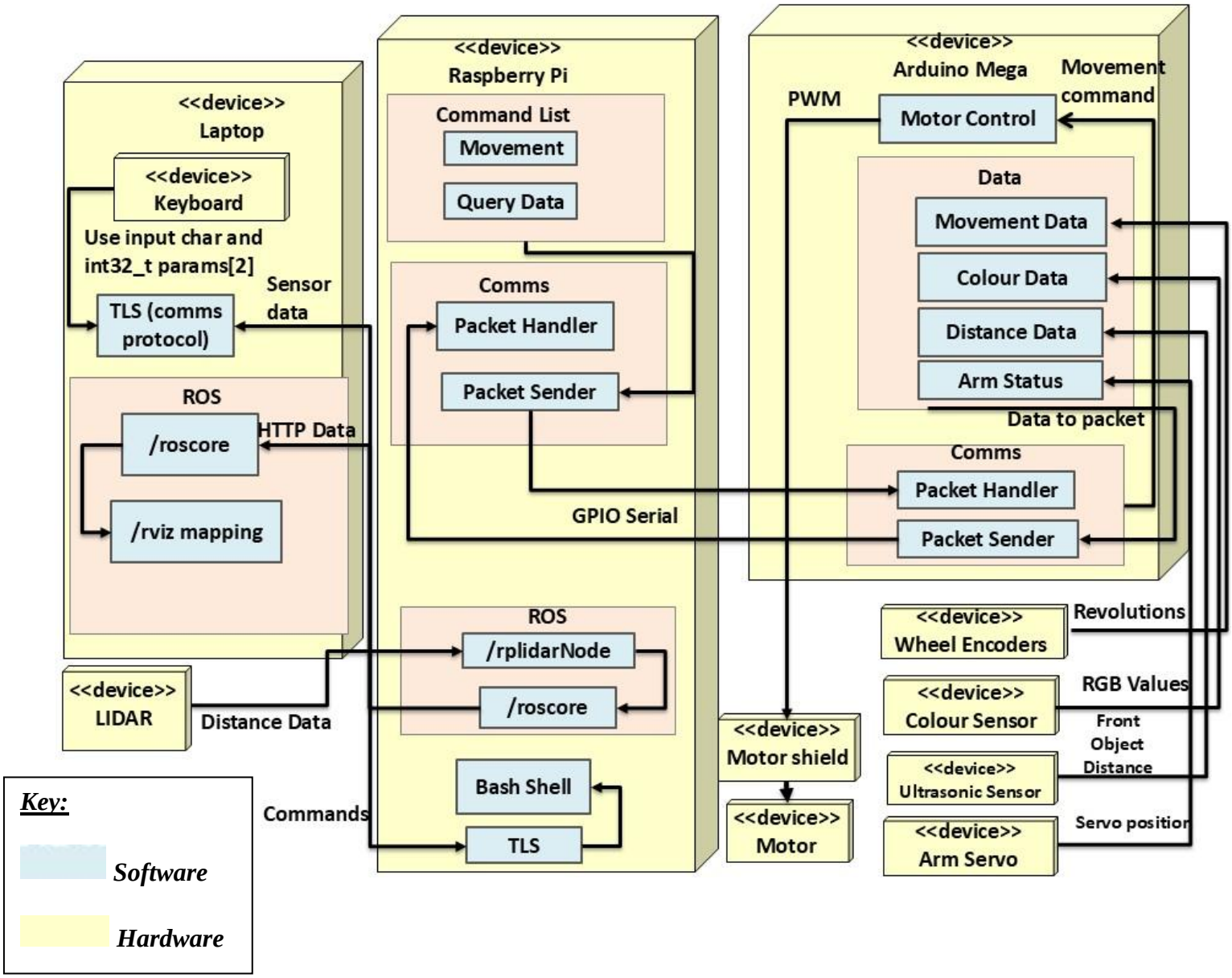
Strengths	***Weaknesses***
Compact and Rugged: Can fit into tight spaces and take a beating.	**Low Autonomy:** Primarily manually controlled, vulnerable to signal distributions
Rich Sensor Suite: Thermal, gas, and environmental sensors onboard**.**	**No Actuators:** Can't move objects or interact with surroundings.
Dual Communication Modes: Switches between wireless and tethered	**Limited Range:** Range depends on signal strength or tether length.

Earthshaker:

Earthshaker is a ground robot built for search and rescue in urban, hazardous, and rubble-filled environments. It was the winning entry in the first Advanced Technology and Engineering Challenge (A-TEC), demonstrating strong disaster response capabilities. Teleoperation is achieved via a tri-modal communication setup between the Operator PC and the onboard Intel NUC, using a 1.8 GHz MIMO mesh radio, two AT9S transmitters, and a 4G/5G router. Its software enables automatic switching between modes based on signal quality. A STM32 control board handles low-level communication between sensors and actuators, while the system is managed remotely through a standard operator interface. Earthshaker includes a front dozer blade for moving obstacles up to 75 kg and features IP64-rated protection for wet conditions (Zhang, 2023).

Strengths	***Weaknesses***
Water-resistant (IP64)	Battery lasts only 3 hours
Supports multiple communication modes	Struggles with vertical axis navigation
Dozer blade can push objects up to 75 kg	Large size limits movement in tight spaces

Section 3 System Architecture



Section 4 Component Design

High-Level Algorithm Steps:

1. Initialize Communication between Arduino Mega, Raspberry Pi, and Laptop
2. Generate map with SLAM
3. Receive tele-operator commands
4. Execute commands (including claw movements)
5. Repeat steps 3 and 4 until the mission is completed

Step 1: Initialize Communication between Arduino Mega, Raspberry Pi, and Laptop

1. Arduino Mega – RPi Communication
    1.1. USB Serial Communication between RPi and Arduino Mega.
    1.2. RPi creates, serializes and sends a Hello Packet to the Arduino Mega.
    1.3. Arduino Mega initializes the serial port and responds to the Hello Packet with an OK packet.
2. RPi – Laptop Communication
    2.1. RPi opens a TCP/IP server and binds to a specific IP address and port.
    2.2. The laptop connects to the server via the same hotspot or WiFi network.
    2.3. The laptop authenticates the connection with the RPi using OpenSSL TLS encryption and ensures secure communication.

Step 2: Generate map with SLAM

1. RPi obtains distance data from LIDAR and odometry data from Arduino Mega.
2. RPi sends occupancy data to the laptop through ROS.
3. Laptop generates map through SLAM using odometry data.

Step 3: Receive tele-operator commands

1. After the map generated has been received, the tele-operator sends any of the following commands

Letter	**W**	**A**	**D**	**S**	**T**
Command	Forward (Short Range)	Left	Right	Reverse (Short Range)	E-Stop

Letter	**P**	**L**
Command	Forward (Long Range)	Reverse (Long Range)

Letter	**X**	**M**	**U**	**C**	**G**	**Q**
Command	Colour Detection	Control Arm	Front Distance	Clear Stats	Get Stats	Quit

2. The PC sends commands to the Pi over TCP/IP, with TLS encryption, using the packet sender.
3. The Pi processes the encrypted packets and forwards the commands to the Arduino.

Step 4: Execute commands

a. Depending on the command, Arduino does the following:

Command Type	Movement	Colour	Ultrasonic Sensor	The Claw
Execution	Generates PWM signals to control motor speed and direction	Activates the color sensor to initiate detection	Activates the ultrasonic sensor to begin distance measurement	Triggers the Claw
Return	Computes position data using feedback from wheel encoders	Identifies colour based on analysed RGB sensor readings	Provides measured distance from ultrasonic sensor	Grips the Object (Astronaut) and returns grip status

b. Raspberry Pi receives the data from the Arduino and LIDAR and handles it as follows:

Arduino		**LIDAR**
Colour	*Ultrasonic*	*Mapping Data*
Returns data to PC through TCP/IP with TLS encryption		Publishes to ROS topic

c. Laptop receives data and does the following:

TLS data	ROS data
Displayed to Tele-Operator	Updated SLAM map is displayed to tele operator

Step 5: Repeat steps 3 and 4 until the mission is completed

Serial Communication

Raspberry PI (TLS Server):

1. Pi establishes a Serial Connection with Arduino Mega with the following configuration:
    - Baud Rate: 9600bps
    - Frame Format: 8N1, 8 data bits, No parity, 1 stop bit

    This is done through startSerial(PORT_NAME, BAUD_RATE, 8, 'N', 1, 5); function. The '5' here refers to the maximum attempts the RPi tries, before declaring that it is unable to open a serial port.

2. Pi creates a receiver thread to begin communication with Arduino
    - A dedicated thread (uartReceiveThread) is created using "pthread_create()" function to continuously read from the serial port
    - Through this, the thread function uartReceiveThread reads incoming bytes from the serial port.
    - It then deserialises into a TPacket packet and a TResult result.
    - If the result received is PACKET_OK, it processes the packet using a separate handler function, handleUARTPacket.
    - Based on the packetType, different handlers are invoked:
        - Response Packets → handleResponse()
        - Error Packets → handleErrorResponse()

    - Message Packets → handleMessage()
  - Otherwise, if the packet received has errors, such as bad magic number or bad checksum, it sends error messages to the remote client on the PC using TLS.

  The receiver thread is crucial for receiving information on the color detected by the Color Sensor and the distance recorded by the Ultrasonic Sensor from the Arduino.

3. Pi sends command packets to Arduino based on the teleoperator commands
   - A TPacket commandPacket is created.
   - Based on the teleoperator command sent through TLS, the "command" field of the commandPacket is configured accordingly.
   - Using serialize() function, the commandPacket is serialized into a TComms packet in "serialize.cpp" file.
   - This is then sent over to the Arduino through UART.

   This is crucial to send commands to the Arduino, so the robot can execute the teleoperator commands.

Arduino Mega (ATmega2560):

1. Arduino establishes a Serial Port for the Pi, in the following configuration
   - Baud Rate: 9600bps
   - Frame Format: 8N1, 8 data bits, No parity, 1 stop bit

2. Arduino Polls for Serial Data
   - Using readPacket(), readSerial() is called to deserialize any serial data received from the Pi.
   - This is stored in TResult result and TPacket packet.

3. Packet Handling
   - Here, Arduino checks the result to determine if the packet is valid or invalid.
   - If the packet is **valid** (PACKET_OK), Arduino handles and processes the packet, performs the corresponding action and sends an acknowledgement, sendOK().
   - If the packet is **invalid**, due to Bad Packet or Bad Checksum, Arduino responds with appropriate functions to handle the error packets (eg: sendBadPacket(), sendBadCheckSum()).

4. Packet Creation
   - This is done in a similar way to Pi, using the serialize() function.
   - For Arduino, we do so using the sendResponse() function, which:
     - receives the address of TPacket.
     - serialises the packet to a TComms format.
   - The TComms packet is then sent over the serial port to the Pi.

Color detection using TCS230

1. Set pins S0, S1, S2 & S3 to OUTPUT and pin OUT to INPUT to register light frequency
2. Set S0 and S1 to HIGH and LOW respectively to produce output frequency scaling of 20% (ideal for arduino)
3. Toggle state of S2 and S3 to produce Red, Green and Blue photodiode lighting based on *table 1*.
   a. For each photodiode type toggled, take the average of 20 frequency readings.

4. By testing and comparing the different frequency values obtained, the colour of the object in front of Alex is determined, and the information will be relayed to the R Pi and then to the server/laptop.

S0	S1	Output Frequency Scaling
L	L	Power down
L	H	2%
H	L	20%
H	H	100%

S2	S3	Photodiode Type
L	L	Red
L	H	Blue
H	L	Clear (no filter)
H	H	Green

Table 1: TCS230 S0:S4 pins and configurations

The Claw

1. The Arduino Mega 2560 will use its 16-bit Timer 5 (TCCR5x, OCR5x) to generate phase correct PWM signals with a prescaler of 8 for controlling the claw servos.
2. Two servo motors will be connected to PWM pins 45 (PL4/OC5B) and 46. (PL3/OC5A). Two different sets of duty cycle values (pulse widths) will be used to achieve the open and close orientations required to grasp and release the astronauts.
3. The claw state (open/close) will be toggled by the "M" command, relayed from the client/laptop to the Raspberry Pi and then to the Arduino. The specific duty cycle values for open and closed will be determined through calibration.

Ultrasonic Distance Sensing

1. To send the pulse, set the trigger pin to high, for approximately 10 microseconds.
2. Using the Echo Pin, calculate the time taken to receive back the pulse.
3. Using the formula below, we can calculate the distance a wall is from the robot:

$$Distance = \frac{Speed\ of\ sound\ \times TIme\ Taken}{2}$$

4. This distance information is sent from the Arduino to the RPi using a packet.

Additionally, we will enable interrupts for the Ultrasonic Sensor when Alex travels long distances using the "P" command, ensuring the robot detects obstacles in real time and avoids collisions with walls or astronauts.

Movement

Overview

For forward and reverse maneuvers, command "W" and "S" will be used to travel 4cm forward and backwards respectively for minute positional adjustments, while "P" and "L" is used for travelling longer distances (12cm). "A" and "D" will be set to turning 15 degrees for every command called.

Distance

The two motors at the front of the robot will be equipped with the encoders, which will track the distance travelled by the robot. For each revolution, the encoder will generate a specific number of ticks. By measuring the circumference of the wheel, we can calculate how many ticks correspond to that distance.

Using this information, we can create a function that converts the required distance into the corresponding number of ticks. The function will count up to the target ticks, and once the target is reached, it will stop, indicating that the desired distance has been traveled. We can calculate the target ticks, using the following formula:

$$Target\ ticks = \left(\frac{Distance\ traveled}{Wheel\ Circumference}\right) \times Ticks\ per\ revolution$$

Angle

Assuming that Alex remains at the same vector space when rotating, A measure of its turning circumference is taken. The total number of wheel turns for one revolution of Alex can then be calculated from (circumference/wheel_circumference). Thus to turn any arbitrary angle, the number of wheel turns would be (angle/360.0 * Alex_circ/wheel_circ).
With that, a separate variable named "deltaTicks" measured below:

$$delta\ Ticks = \frac{Arc\ of\ circle\ to\ turn}{Wheel\ circumference} \times Ticks\ per\ revolution$$

$$= \frac{(\frac{intended\ angle}{360} \times cicumference)}{Wheel\ circumference} \times Ticks\ per\ revolution$$

$$= \left(\frac{intended\ angle}{360}\right) \times \left(\frac{Circumference}{Wheel\ circumference}\right) \times Ticks\ per\ revolution$$

As such, if Alex has a left/right reverse ticks turn of zero, Alex would rotate until the number of left/right reverse ticks increments to the amount of ticks in deltaTicks calculated.

Section 5 Project Plan

Week	Hardware	Software
Week 10	Initial robot build (chassis, motors, sensors), begin arm design & wiring	Set up UART, implement basic Teleop, start learning ROS + SLAM
Week 11	Focus on robotic arm: mounting, servo tuning, wiring neatness	Refine Teleop accuracy (turns, speed), integrate ROS with LIDAR an
Week 12	Finalize full assembly, secure components and wiring	Calibrate sensors, fine-tune SLAM, work on robotic claw control with dummy astronauts
Week 13	Trial runs, fix hardware issues, final adjustments	Final software cleanup, validate mapping/control, prep for evaluation
Reading Week	Submit final report and documentation	

References

Silva, J., Almeida, L., & Lima, P. (2006). RAPOSA: Semi-Autonomous Robot for Rescue Operations.

Zhang, J. (2023). Earthshaker: A mobile rescue robot for emergencies and disasters through teleoperation and autonomous navigation. *JUSTC*, 53.