Team L102 (Muskrat)
Software documentation

CONCEPTS AND DESIGN DECISIONS

The initial idea of the project was using computer vision to locate the robot and victims. Connecting the overhead camera to a laptop, we would use the OpenCV2 library to get a 480x640x3 image array and obtain x-y centres of the 4 green objects (victims), of a red star placed above the grabber, and a blue star above the rotational centre of the robot. These 6 pairs of x-y coordinates would then be transmitted over wifi for the Aruduino to decide how to act.
The coordinates of the red and blue stars would give the position and direction vectors of the robot, and in each cycle it would try to align with a victim and move forwards proportionally to the distance needed to be covered (up to 3 seconds). After stopping, the robot would request a new information package and repeat the cycle until reaching a victim.

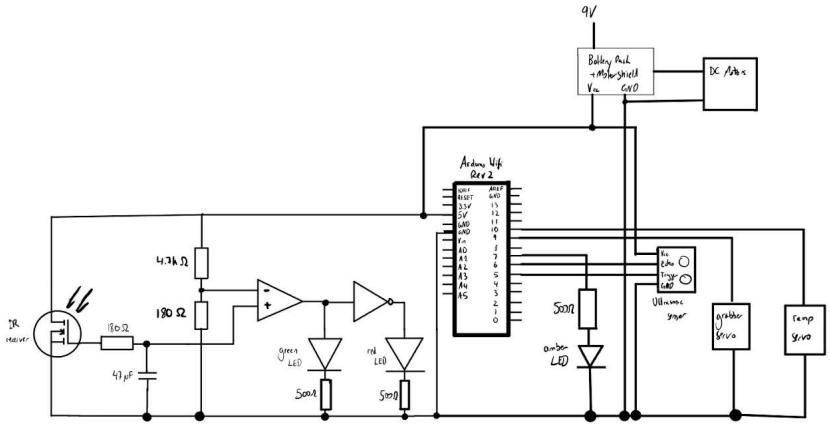
With only a couple of days before the competition, this idea was radically changed as we discovered a lag between the movement of the robot and data transmission. Concretely, we had a function which would rotate the robot 180 degrees (when aligning with the victim), but because the commands received were one cycle behind, the robot would enter an infinite loop and would spin in place. This, combined with the whole data processing and transmission being relatively slow, made us decide to hard-code movement in and out of the tunnel (and to triage and finish areas), and locate victims by rotating and sweeping the 'cave' using an ultrasound sensor. Then, the robot would travel the measured distance, pick up the victim and return backwards following the same path, rotate to the initial position and exit the tunnel backwards.

SYSTEM DIAGRAM

Physical connections:

Components on the shield board connected to the Arduino:

- D2 – IR sensor input
- D3 – green LED
- D4 – red LED
- D5 – ultrasound trigger
- D6 – ultrasound echo
- D7 – amber LED
- D9 – grabber servo
- D10 – ramp servo



Serial connections between Arduino and PC

Despite being finally deprecated, our protocol for data transfer was telnet since it is based on the reliable TCP protocol. Though slower than the alternative option of UDP, it safeguards against problems caused by lost data packages, which seemed more important. Using the Arduino's wifi module and the python library telnetlib, we could then set up the Arduino as a server to send and receive streams of 8-bit characters to and from a laptop. Bit manipulation allowed the 12 coordinates (4 for robot, 8 for victims, each encapsulated by 10 bits) to be converted into a stream of 15x8bit characters for transfer and then vice versa on the arduino side.

Showing victim detection

The amber LED is blinking when the robot is moving, and stays turned on when the victim is picked up. Additionally, when the ultrasonic sensor makes a detection closer than a certain threshold, the robot stops rotating and starts moving forward, which can be used to signal that a victim has been targeted. Finally, green (healthy) and red (injured) LED's show the state of the victim once it is approached enough.

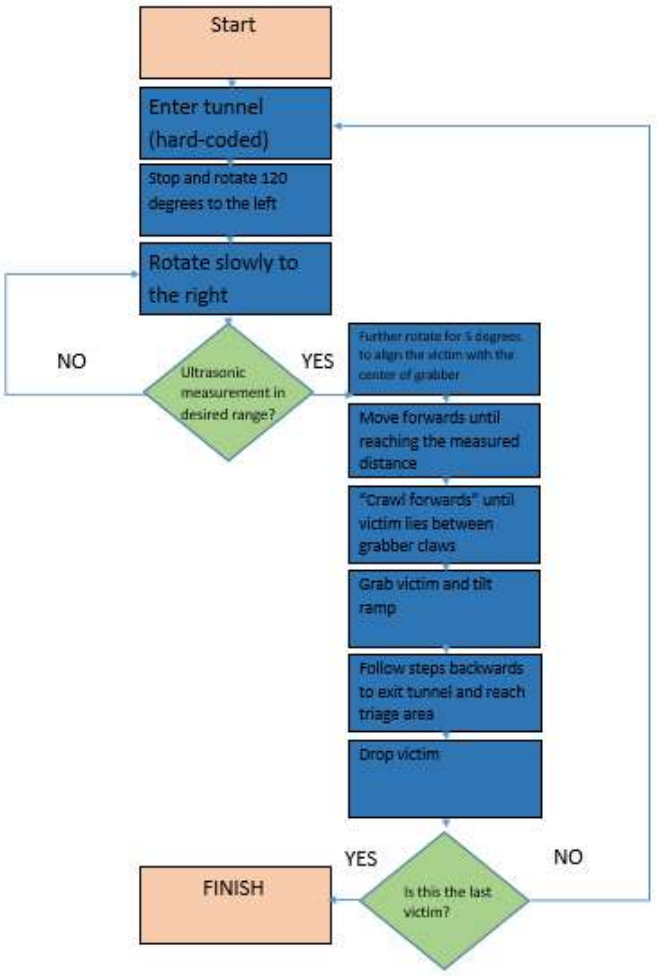
Identifying position

This includes two different problems: identifying the victims and the robot. As soon as the robot enters the 'cave' hardcoded, it slowly rotates from -120 to 120 degrees, and takes requests a measurement from the ultrasound sensor every 20 milliseconds. If the value is closer than 2 m and at least 30 cm closer than the previous measurement (to make a distinction between victims and walls) a victim is considered to be detected. Regarding the robot, there is no exact control over its absolute position. Instead, every movement back is done by following the same paths backwards (with the same speed and for equal duration).

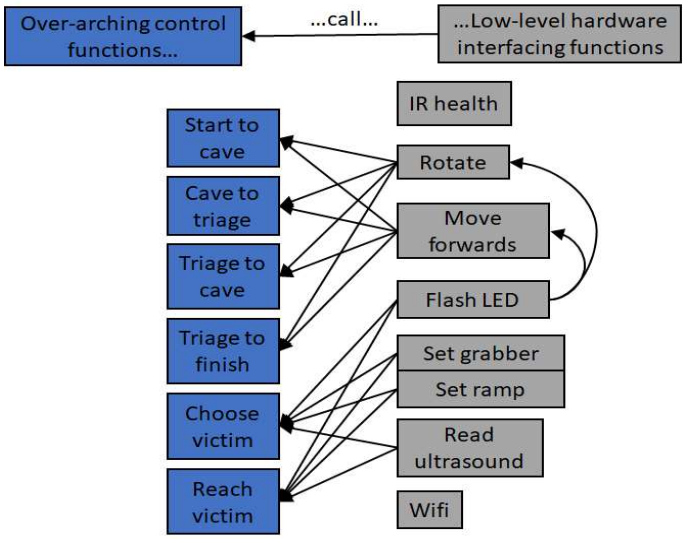
The initial method used image processing to get the coordinates of the robot and victims. The processing involved first masking out fixed unwanted areas of the image such as the tunnel or table surroundings. It then reduced the RGB components of each pixel down so that one colour component remained (to account for local lighting), and then applied OpenCV thresholding and contouring with perimeter and enclosed-area conditions to identify the victims and coloured stars. Despite providing accurate locationing, this method used a manually written filter instead of solely using the OpenCV python wrappers and so was unsuitably slow, causing errors to propagate much longer before correction. Possible use of OpenCV's conversion into HSL space for analysis may have allowed for the same local thresholding whilst only using OpenCV's python wrappers.

CODE STRUCTURE AND ALGORITHMS

Overall strategy



Function Interaction



Structure used

The code is structured such that the main loop only decides the overall strategic order of the mission actions, by calling or "handing over control" to each of the overarching control functions, each with their own conditional loop. Global flags are used to inform the main loop which actions are completed.

Each of these overarching functions carries out actions using the low-level hardware interaction functions, so that action specific details are contained as much as possible. Exceptions to note are when the victim related navigation has movements depending on the ultrasonic readings, to which we decided to alter and embed the movement functions.

Other points to note are that the IR health detection works as a standalone from the software, that the wifi was eventually not used, and that the LED flashing is used within the normal movement functions.

Source code: https://github.com/SamuelmsWong/Muskrat